Mapa Denso da Disparidade sobre a Imagem Estereoscópica Colorida no Domínio da Álgebra de Clifford

André Luiz Pasquali

Mapa Denso da Disparidade sobre a Imagem Estereoscópica Colorida no Domínio da Álgebra de Clifford

André Luiz Pasquali

Dissertação de Mestrado submetida à banca examinadora designada pelo Colegiado do Programa de Mestrado em Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul, como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 29 de novembro de 2010 por:

Milton Romero Romero - Dr.
Prof. DEL/UFMS - Orientador

Hemerson Pistori - Dr.
Prof. CCET/UCDB

Evandro Mazina Martins - Dr.
Prof. DEL/UFMS

Agradecimentos

Agradeço:

Primeiramente a Deus, por tudo que eu consegui realizar durante todos esses anos.

Aos meus pais José Carlos Pasquali e Bernardete Pasquali pelo incentivo e apoio durante o curso e em todas as etapas de minha vida.

Ao meu orientador professor Dr. Milton Romero Romero e o professor Dr. Evandro Mazina, que foram pessoas importantes para o meu desenvolvimento pessoal e profissional nesses anos todos, pela paciência e dedicação demonstrados, sempre me ajudando e auxiliando no desenvolvimento dos estudos.

Aos colegas e professores do mestrado de Engenharia Elétrica.

Ao pessoal do CTEIMS, a Patricia, o Mario, o André, o Guilherme e o Fabrício, que estiveram sempre comigo e acompanharam o desenvolvimento do meu trabalho.

A minha namorada Gabriela, que tem me acompanhado nessa jornada e pacientemente tem suportado minhas inquietudes.

"Faça as coisas o mais simples que você puder,
porém não as mais simples"

Albert Einstein

Resumo

Extensões do Quad-tree para a estimação do mapa denso da disparidade utilizando a luminância no domínio real já foram propostas na literatura. Neste trabalho o Quad-tree será estendido para processar imagens coloridas (RGB e YUV) codificadas no domínio da álgebra de Clifford. O mapa denso da disparidade é obtido através do cálculo da correspondência para cada pixel entre duas imagens estereoscópicas. O cálculo da correspondência é uma tarefa fundamental que apresenta problemas devido à presença de áreas oclusas, ruídos e variações na iluminação, entre outros. A álgebra de Clifford é utilizada para mapear os valores de cor de cada pixel nos vetores de base, compondo assim um número de Clifford. No domínio de Clifford proposto, a extensão do Quad-tree gera resultados melhores para imagens mais complexas se comparados com os obtidos sobre as imagens utilizando o domínio real sobre a luminância. Processando vários pares estereoscópicos no domínio real (luminância) e no domínio de Clifford (cores) se demonstra experimentalmente a melhora obtida para imagens em cenas reais.

Abstract

Extensions of the Quad-tree for the estimate dense disparity map using the luminance on the real domain have been proposed in the literature. In this work the Quad-tree is extended to process color images (YUV and RGB) encoded in the field of Clifford algebras. The dense disparity map is obtained by performing the correlation for each pixel between two stereoscopic images. The correspondence problem is an essential task that presents problems due to the presence of occluded areas, noise and variations in lighting, among others. The Clifford algebra is used to map the color values of each pixel in the basis vectors, represented by a single Clifford number. In the Clifford domain the proposed extension of the Quad-tree yield better results for more complex images as compared with those obtained on the images using the real domain over the luminance. By processing several stereoscopic pairs on the real domain (luminance) and in the Clifford domain (color) experimentally the improvement obtained for stereoscopic images in real scenes is demonstrated.

Sumário

Resumo **i**

Abstract **ii**

1 Introdução **1**

2 Estereoscopia **4**

3 Modelos de cor RGB e YUV **8**
3.1 Modelo RGB 8
3.2 Modelo YUV 9

4 Quad-Tree **10**

5 Álgebra de Clifford **15**
5.1 Soma Geométrica 15
5.2 Produto Geométrico 16
5.3 Produto Interior 16
5.4 Produto Exterior 17
5.5 Mapeamento das cores 18
5.6 Convolução em Clifford 18
5.7 Norma de Clifford 19
5.8 Mean Absolute Difference em Clifford 19

6 Extensão do Quad-Tree **20**

7 Resultados e Discussão dos Resultados **22**

8 Conclusões **42**

Referências **45**

A Códigos Fonte Estereoscopia em Clifford **46**

B Códigos Fonte Erro Quadrático Médio **65**

Lista de Figuras

1.1 Par estereoscópico do trem. 1

2.1 Exemplo de área de busca completa. 5
2.2 Plano de projeção da disparidade. 6
2.3 Imagem esquerda do trem. 6
2.4 Mapa denso da disparidade do par estereoscópico da Figura (1.1). 7

3.1 Cubo RGB. 9

4.1 Estrutura do Quad-tree. 11
4.2 Filtro antialiasing (Tabela (4.1)). 12
4.3 Expansão de cada pixel do mapa denso da disparidade do nível K ao $K+1$ (*Upsampling*). 13
4.4 Sequência de passos para atualização e interpolação do mapa denso da disparidade. 13

5.1 Mapeamento dos componentes RGB nos vetores de base (e_1, e_2, e_3). 19

6.1 Quad-Tree estendido no domínio de Clifford. 21

7.1 Par estereoscópico do carrossel. 23
7.2 Resultados sobre a Luminância das imagens do carrossel 24
7.3 Resultados de Clifford com RGB das imagens do carrossel 24
7.4 Resultados de Clifford com YUV das imagens do carrossel 24
7.5 Resultados de Clifford com YUV2 das imagens do carrossel 25
7.6 Par estereoscópico do trem. 26
7.7 Resultados sobre a Luminância das imagens do trem. 27
7.8 Resultados de Clifford com RGB das imagens do trem para cena 1. 27
7.9 Resultados de Clifford com RGB das imagens do trem para cena 2. 27
7.10 Resultados de Clifford com YUV das imagens do trem para cena 1. 28
7.11 Resultados de Clifford com YUV das imagens do trem para cena 2. 28
7.12 Resultados de Clifford com YUV2 das imagens do trem para cena 1. 28
7.13 Resultados de Clifford com YUV2 das imagens do trem para cena 2. 29
7.14 Par estereoscópico do rinoceronte. 30
7.15 Resultados sobre a Luminância das imagens do rinoceronte. 30
7.16 Resultados de Clifford com RGB das imagens do rinoceronte. 31
7.17 Resultados de Clifford com YUV das imagens do rinoceronte. 31
7.18 Resultados de Clifford com YUV2 das imagens do rinoceronte. 31
7.19 Par estereoscópico do flamingo. 33

7.20 Resultados sobre a Luminância das imagens do flamingo. 33
7.21 Resultados de Clifford com RGB das imagens do flamingo. 34
7.22 Resultados de Clifford com YUV das imagens do flamingo. 34
7.23 Resultados de Clifford com YUV2 das imagens do flamingo. 34
7.24 Par estereoscópico dos peixes. 36
7.25 Resultados sobre a Luminância das imagens dos peixes. 36
7.26 Resultados de Clifford com RGB das imagens dos peixes. 37
7.27 Resultados de Clifford com YUV das imagens dos peixes. 37
7.28 Resultados de Clifford com YUV2 das imagens dos peixes. 37
7.29 Par estereoscópico do riacho. 39
7.30 Resultados sobre a Luminância das imagens do riacho. 39
7.31 Resultados de Clifford com RGB das imagens do riacho. 40
7.32 Resultados de Clifford com YUV das imagens do riacho. 40
7.33 Resultados de Clifford com YUV2 das imagens do riacho. 40

Lista de Tabelas

4.1 Coeficientes do filtro *antialiasing*. 12
4.2 Resposta impulso filtro LPF_K. 12

5.1 Produto geométrico. 17
5.2 Produto interior. 17
5.3 Produto exterior. 18

7.1 Resultados do SNR obtidos das imagens do carrossel. 25
7.2 Resultados do SNR obtidos das imagens do trem. 29
7.3 Resultados do SNR obtidos das imagens do rinoceronte. 32
7.4 Resultados do SNR obtidos das imagens do flamingo. 35
7.5 Resultados do SNR obtidos das imagens dos peixes. 38
7.6 Resultados do SNR obtidos das imagens do riacho. 41

CAPÍTULO

1

Introdução

A estereoscopia é utilizada em diversas aplicações que envolvem visão em profundidade, tais como: a visão de robôs, a realidade virtual, a realidade virtualizada, TV 3D (Chiari et al., 1997), etc. A estereoscopia possibilita a reconstrução de estruturas tridimensionais através de duas imagens de uma mesma cena, mas de ângulos ligeiramente defasados, como mostra o par estereoscópico da Figura (1.1).

(a) Imagem esquerda. (b) Imagem direita.

Figura 1.1: Par estereoscópico do trem.

O cálculo do mapa denso da disparidade consiste em encontrar a correspondência para todos os pixels da imagem esquerda em relação à direita ou vice versa, e apresenta problemas causados por regiões oclusas, ruídos e variações na iluminação das imagens utilizadas, diferentes ganhos das câmeras, etc.. Os pixels correspondentes são aqueles que estão presentes tanto na imagem esquerda quanto na direita. Essas reconstruções são possíveis porque utilizando o

par estereoscópico podemos estimar o deslocamento ou disparidade de cada pixel pertencente à imagem esquerda em relação à direita ou vice versa.

Várias abordagens têm sido utilizadas para o cálculo da disparidade como correspondência de blocos (Brown et al., 2003; Banks e Corke, 2001), otimização de gradiente (Brown et al., 2003), correspondência de características (Dhond e Aggarwal, 1989) e Quad-tree (Chiari et al., 1997), entre outros.

Em Chiari (Chiari et al., 1997), o algoritmo Quad-tree de análise multirresolução, toma a luminância de cada imagem, esquerda e direita, à plena resolução de $720x576$ e aplica uma convolução com o filtro antialiasing. Após cada imagem ser filtrada é realizada uma dizimação por 2 gerando dois novos ícones na resolução de $360x288$, tanto a filtragem quanto a dizimação são repetidas até que sejam produzidos dois ícones de $45x36$, um referente a imagem esquerda e outro a direita. Com o par de ícones esquerdo e direito de $45x36$ e utilizando um suporte de $9x9$ pixels centrados sob o pixel em consideração, é calculada a disparidade de cada pixel gerando assim o primeiro mapa denso da disparidade de $45x36$. Posteriormente, se aumenta a resolução através da multiplicação por 2 da disparidade do mapa de $45x36$ e se duplica a posição gerando um novo mapa com valores conhecidos nas posições pares e valores desconhecidos nas impares. O processo seguinte é atualizar e interpolar esses valores para completar o mapa denso da disparidade na resolução de $90x72$. Repete-se este processo até se obter o mapa a plena resolução de $720x576$.

Neste trabalho se estende o anterior algoritmo Quad-tree proposto por Chiari em (Chiari et al., 1997), para trabalhar com imagens coloridas em álgebra de Clifford (Hestenes, 2003; Dorst e Mann, 2002; Mann e Dorst, 2002). O domínio de Clifford de assinatura Cl_3 é isomorfo ao espaço Euclidiano 3D e composto por um escalar 1, 3 vetores e_1, e_2, e_3, 3 bivetores e_{12}, e_{13}, e_{23}, e o pseudoescalar e_{123}. O escalar é uma grandeza adimensional ou número puro, os vetores no domínio de Clifford são análogos ao i, j, e k do espaço Euclidiano, os bivetores são áreas orientadas e o pseudoescalar ou trivetor é um volume orientado. Sobre esse domínio se define o produto geométrico representado pelo símbolo $*$ como será discutido mais a frente. Portanto foi necessário que tanto as imagens como filtro fossem definidos no domínio de Clifford, assim como a convolução e a função de correlação, utilizado no processo de dizimação e cálculo do mapa denso da disparidade.

Para trabalhar com imagens coloridas em Clifford, realizamos três mapeamentos dos valores de cor do pixel em um número de Clifford: no primeiro os valores de RGB de cada pixel foram mapeados nos vetores da álgebra (e_1, e_2, e_3), isto é $P_{x,y} = R_{x,y} * e_1 + G_{x,y} * e_2 + B_{x,y} * e_3$, no segundo os valores de YUV foram mapeados nos vetores da álgebra, assim $P_{x,y} = Y_{x,y} * e_1 + U_{x,y} * e_2 + V_{x,y} * e_3$, no terceiro o valor de luminância Y foi mapeado no escalar (1) e as crominâncias U e V nos vetores, assim $P_{x,y} = Y_{x,y} * 1 + U_{x,y} * e_1 + V_{x,y} * e_2$, x e y são as coordenadas do pixel $P_{(x,y)}$.

As imagens usadas no cálculo do mapa denso da disparidade são as mesmas utilizadas no trabalho de Chiari (Chiari et al., 1997), para podermos fazer uma comparação entre os resultados

do processamento a cores no domínio de Clifford e o processamento sobre a luminância no domínio dos reais. Cada imagem do par estéreo tem resolução de $720x576$ pixels, utilizada na TV 3D, 25 pares por segundo no sistema europeu.

Os resultados obtidos com o Quad-tree estendido, no domínio de Clifford, comparados com os resultados apresentados em Chiari (Chiari et al., 1997), obtiveram tanto uma maior relação sinal/ruído (SNR), quanto um melhor resultado visível sobre as imagens mais complexas. Nas imagens mais simples foram obtidos resultados semelhantes. Através da comparação entre o SNR e a análise visual dos resultados foi possível identificar que, utilizando imagens coloridas em Clifford, a relação sinal/ruído é maior o que em alguns casos pode indicar, desta forma, melhor qualidade do mapa gerado, tendo em vista que o SNR nem sempre condiz com a qualidade visível da imagem o que é verificado através da analise visual.

O restante deste trabalho está organizado da seguinte forma: no capítulo 2 é apresentada a estereoscopia; no capítulo 3 apresentamos os modelos de cor com RGB e YUV; no capítulo 4 temos uma descrição detalhada do Quad-tree proposto por Chiari (Chiari et al., 1997); no capítulo 5 apresentamos a álgebra de Clifford; no capítulo 6 temos a extensão proposta do Quad-tree na álgebra de Clifford; no capítulo 7 são apresentados os resultados com a discussão dos resultados; e no capítulo 8 temos as conclusões e trabalhos futuros.

CAPÍTULO

2

Estereoscopia

A estereoscopia, em princípio, possibilita a reconstrução de estruturas tridimensionais através de duas imagens de diferentes perspectivas, imagens estereoscópicas como mostrado na Figura (1.1). Essas reconstruções são realizadas através do mapa de disparidade que possui informações sobre o deslocamento dos pixels da imagem esquerda em relação à direita, ou vice versa. A qualidade das reconstruções depende da determinação dos pixels correspondentes entre as imagens, conhecido na literatura como o problema da correspondência.

Encontrar o pixel correspondente entre duas imagens estéreo é uma tarefa fundamental, pois o pixel pode estar presente apenas em uma das imagens, caso ele pertença a uma área oclusa, ou tenha sofrido alguma interferência causada pela diferença de iluminação entre as imagens estereoscópicas, e até mesmo em casos onde o dispositivo de captura das imagens está descalibrado.

O cálculo do deslocamento na posição do pixel correspondente em duas imagens estereoscópicas é chamado disparidade. Com o cálculo da disparidade de todos os pixels da imagem esquerda em relação à direita é gerado o mapa denso da disparidade.

Os pixels correspondentes são determinados através da função de correlação que determina quanto dois pixels ou duas regiões são semelhantes. Para determinar similaridade entre pixels existem diversas funções de correlação presentes na literatura como: Normalized Cross-Correlation (NCC), Sum of Squared Differences (SSD), Normalized SSD, Rank, Census, Brown Burschka Hager (Brown et al., 2003; Banks e Corke, 2001; Dhond e Aggarwal, 1989; Bhat e Nayar, 1998; Mühlmann et al., 2002). Neste trabalho é utilizada a Mean Absolute Difference (MAD) (Chiari et al., 1997), que é dado pela Equação (2.1), e compara duas regiões de tamanho hxk com deslocamento m e n e armazena o índice de menor valor encontrado de acordo com a Equação (2.2), onde abs presente na Equação (2.1) representa o valor absoluto e min na Equação (2.2) representa o índice de menor valor.

$$d(m,n) = \sum_{h,k} abs(P_R(x+h, y+k) - P_L(x+m+h, y+n+k)) \tag{2.1}$$

$$MDD(dx, dy) = min(d(m,n)) \tag{2.2}$$

A estimação do melhor candidato da imagem esquerda $P_L(x+dx, y+dy)$, com o pixel correspondente da imagem direita $P_R(x,y)$, pode ser obtido usando uma área de busca completa, como pode ser visto na Figura (2.1), que é baseada na comparação de um bloco de NxN centrado em um ponto $P_R(x,y)$ do ícone direito, e um conjunto de blocos também de NxN selecionados dentro de uma área de busca no ícone esquerdo. Utilizando o MAD é realizada a comparação de cada bloco da área de busca do ícone esquerdo $P_L(x+dx, y+dy)$, com o bloco centrado sob o ponto de consideração do ícone direito $P_R(x,y)$, o bloco do ícone esquerdo que obtiver o menor valor é selecionado como sendo correspondente com a ícone direito e seu valores dx e dy representam a disparidade horizontal e vertical, respectivamente.

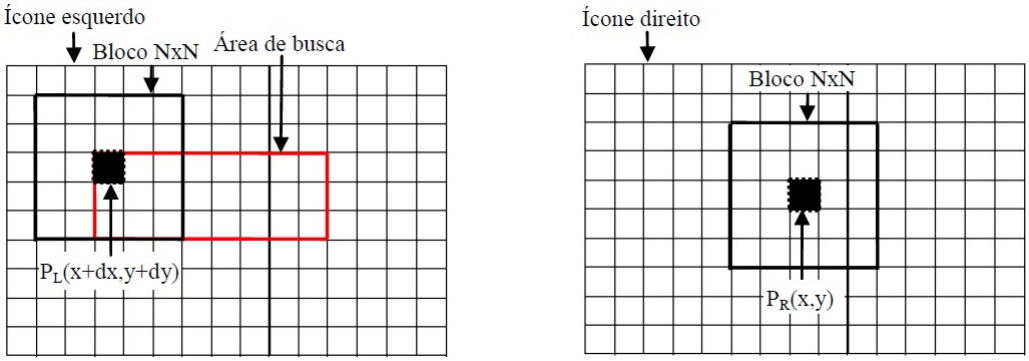


Figura 2.1: Exemplo de área de busca completa.

A disparidade pode ser positiva, negativa ou nula. Como mostra a Figura (2.2), a disparidade é positiva quando o pixel da imagem esquerda é projetado no plano de projeção à direita do seu correspondente da imagem direita. A disparidade é negativa quando o pixel da imagem esquerda é projetado à esquerda do seu correspondente da imagem direita. A disparidade é nula quando o pixel da imagem direita e seu correspondente da imagem esquerda estão projetados no mesmo ponto do plano de projeção. No mapa denso da disparidade os pontos que possuem disparidade positiva, ou seja, que estão mais perto do observador são representados por tons de cinza mais claros, os pontos com disparidade negativa que estão mais ao fundo na cena distante do observador são representados por tons de cinza mais escuros e os pontos com disparidade nula são representados por um tom de cinza mediano, como mostra a Figura (2.4).

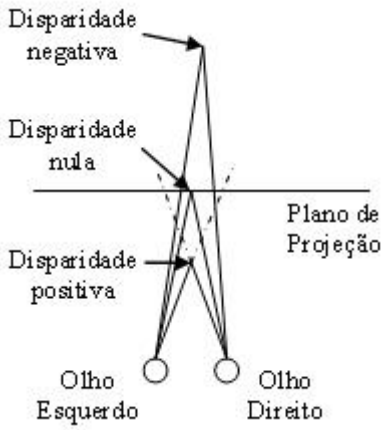


Figura 2.2: Plano de projeção da disparidade.

Figura 2.3: Imagem esquerda do trem.

Figura 2.4: Mapa denso da disparidade do par estereoscópico da Figura (1.1).

CAPÍTULO

3

Modelos de cor RGB e YUV

Os modelos de cor são formas abstratas de representar cores através de tuplas de números, com três ou quatro valores ou componentes de cor. Na literatura existem diversos modelos de cor como: RGB, YUV, HSV e CMYK, entre outros. Neste trabalho vamos utilizar dois modelos de cor o RGB e o YUV, para mapear as informações de cor na álgebra de Clifford.

3.1 Modelo RGB

O modelo RGB é comumente utilizado em monitores e outros dispositivos como câmeras digitais, scanners e datashows. O modelo RGB é composto por 3 cores primarias vermelho (Red), verde (Green) e azul (Blue). Nesse modelo começamos com o preto ou ausência de cores e vamos adicionando diferentes porções de R, G e B para produzir a cor desejada, por isso o modelo RGB é conhecido como um modelo aditivo, onde somadas as intensidades máximas de R, G e B temos a cor branca (Plataniotis e Venetsanopoulos, 2000; Hüllermeier et al., 2010).

As Equações (3.1), (3.2) e (3.3) mostram como fazer a conversão do modelo YUV para RGB:

$$B = 1.164 * (Y - 16) + 2.018 * (U - 128) \tag{3.1}$$

$$G = 1.164 * (Y - 16) - 0.813 * (V - 128) - 0.391 * (U - 128) \tag{3.2}$$

$$R = 1.164 * (Y - 16) + 1.596 * (V - 128) \tag{3.3}$$

Através do sistema de coordenadas do plano cartesiano as cores R, G e B são representados por valores de 0 a 255, a Figura (3.1) apresenta o modelo RGB em um cubo.

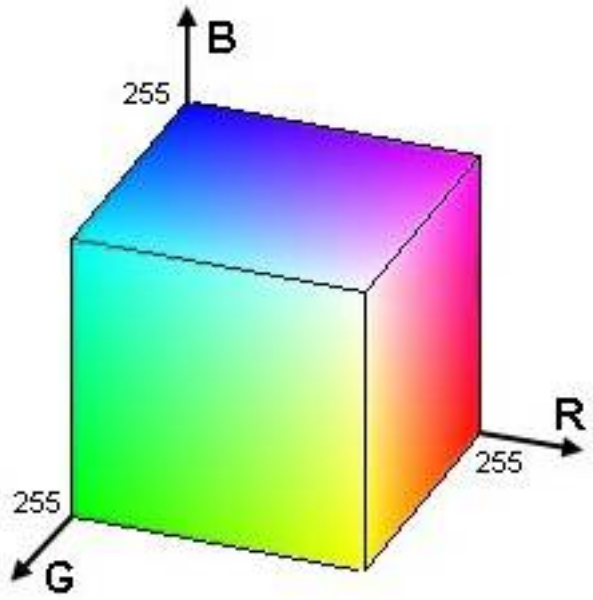


Figura 3.1: Cubo RGB.

3.2 Modelo YUV

O modelo YUV inicialmente foi utilizado na transmissão de sinais para televisão colorida, pois apresenta a luminância Y e duas crominâncias U e V separadamente, o que permite que o mesmo sinal seja compatível tanto para os televisores coloridos quanto para os preto e branco, visto que nas TVs preto e branco se utiliza apenas a luminância.

Y, U e V podem ser obtidos através dos valores de R, G e B, onde Y é uma soma ponderada de R, G e B que representa a luminância, U representa a diferença entre Y e R, V a diferença entre Y e B. A Equação (3.4) apresenta a matriz que permite a mudança de base do modelo RGB para o modelo YUV (Hüllermeier et al., 2010).

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.4}$$

CAPÍTULO

4

Quad-Tree

O Quad-tree é um algoritmo de análise multirresolução. A extensão proposta por Chiari (Chiari et al., 1997) usa diferentes níveis de resolução para calcular a disparidade de um par de imagens estereoscópicas. Este algoritmo possui menor complexidade computacional se comparado a outros algoritmos de busca completa que são utilizados para o cálculo da disparidade das imagens a plena resolução.

A estrutura do Quad-tree presente na Figura (4.1) é descrita a seguir:

1. Balancear a luminância das imagens de entrada, onde é aplicado um filtro de suavização para compensar possíveis desajustes do dispositivo de captura das imagens, onde L e R representam a luminância da imagem esquerda e direita, respectivamente.

2. Aplicar o filtro *antialiasing*, onde é realizada uma convolução entre a imagem e o filtro presente na Tabela (4.1) e na Figura (4.2), que apresenta os 25 coeficientes do filtro utilizados no processo. Como o filtro utilizado é linearmente separável e a imagem é bidimensional o filtro pode ser aplicado em dois passos, inicialmente na horizontal e posteriormente na vertical. O filtro LPF_2 da Tabela (4.2), produz uma versão filtrada da imagem original, que pode ser dizimada por 2 na horizontal e vertical sem a introdução de *aliasing* na imagem resultante (Oppenheim et al., 1999).

3. Downsampling, dizimar a imagem por 2. Nesse passo é criado um ícone, com metade da resolução da imagem filtrada, formado apenas com os pixels pares da imagem.

4. Repetir os passos 2 e 3 até que a imagem tenha a resolução de $45x36$.

5. Calcular o mapa denso da disparidade, MDD, utilizando o MAD, Equações (2.1) e (2.2). Com o ícone de $45x36$ é criado o primeiro MDD também de $45x36$ que será utilizado na fase de *upsampling*.

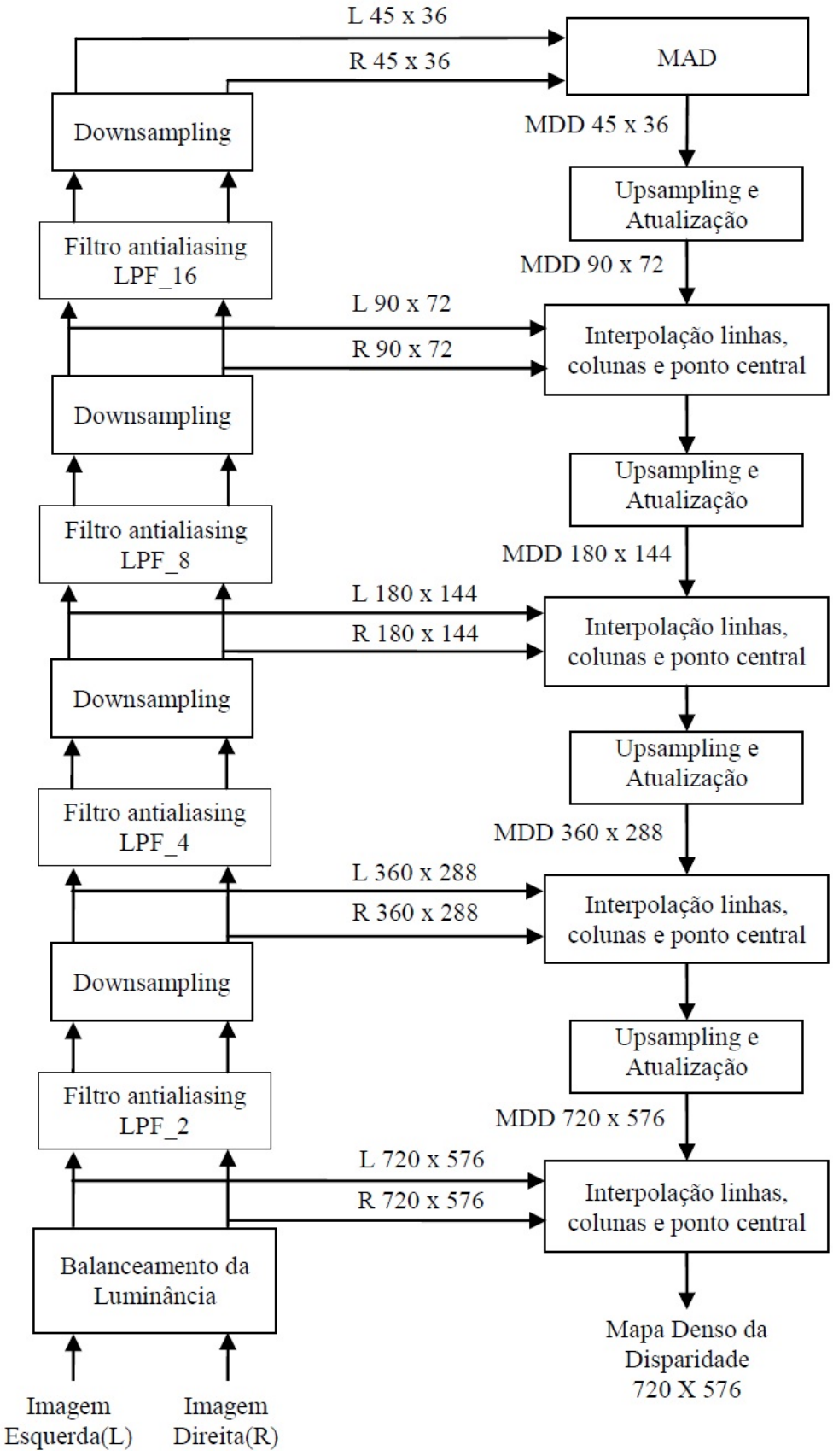


Figura 4.1: Estrutura do Quad-tree.

Coeficientes	Valor	Coeficientes	Valor
$h0$	-0.005615	$h13$	0.314697
$h1$	-0.005127	$h14$	0.029785
$h2$	0.011230	$h15$	-0.095703
$h3$	0.012451	$h16$	-0.025638
$h4$	-0.017090	$h17$	0.047119
$h5$	-0.023193	$h18$	0.021484
$h6$	0.021484	$h19$	-0.023193
$h7$	0.047119	$h20$	-0.017090
$h8$	-0.025635	$h21$	0.012451
$h9$	-0.095703	$h22$	0.011230
$h10$	0.029785	$h23$	-0.005127
$h11$	0.314697	$h24$	-0.005615
$h12$	0.471191		

Tabela 4.1: Coeficientes do filtro *antialiasing*.

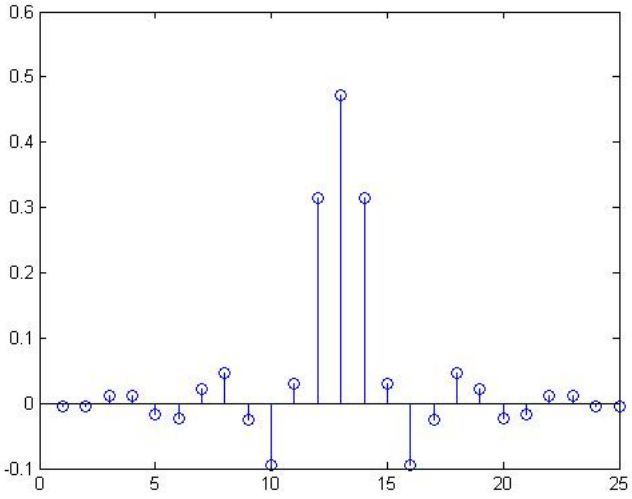


Figura 4.2: Filtro antialiasing (Tabela (4.1)).

Filtro	Função de filtragem
LPF_2	$h2(n) = h(n)$
LPF_4	$h4(n) = h(n) * h(n)$
LPF_8	$h8(n) = h(n) * h(n) * h(n)$
LPF_16	$h16(n) = h(n) * h(n) * h(n) * h(n)$

Tabela 4.2: Resposta impulso filtro LPF_K.

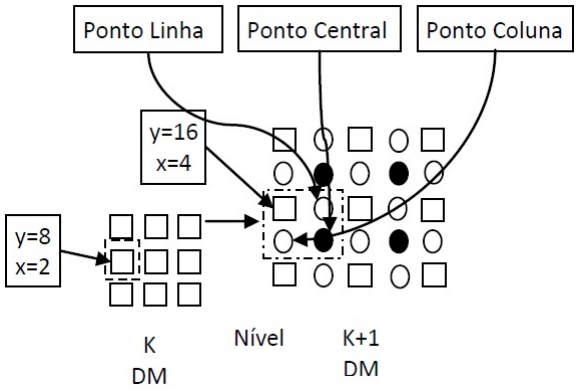


Figura 4.3: Expansão de cada pixel do mapa denso da disparidade do nível K ao $K+1$ (*Upsampling*).

6. Upsampling por 2 na resolução e nos valores do MDD. A Figura (4.3), mostra como é realizada a expansão de cada pixel onde os quadrados representam os pontos com disparidade conhecida e os círculos os pontos com disparidade desconhecida, dentro de um bloco de $2x2$ do nível K, onde o valor da disparidade presente na posição $y = 8$, $x = 2$ é multiplicado por dois e armazenado na posição $y = 16$, $x = 4$ no nível $K + 1$.

Figura 4.4: Sequência de passos para atualização e interpolação do mapa denso da disparidade.

7. Atualização da disparidade das posições pares do MDD, passo 1, Figura (4.4). Utilizando a disparidade das posições pares e com os ícones de luminância direito e esquerdo de $90x72$, é reestimada a disparidade aplicando a função de correlação MAD sobre a vizinhança de $3x3$ de cada pixel com disparidade conhecida.

8. A interpolação do ponto central, das linhas e colunas é realizada na seguinte ordem:

   (a) A disparidade do ponto central (círculos escuros na Figura (4.3)), é estimada em dois passos: primeiro é calculado o baricentro dos seus quatro vizinhos conhecidos (quadrados na Figura (4.3)), e depois a disparidade do ponto central recebe o índice de menor MAD dos 5 pontos, o baricentro e os 4 vizinhos conhecidos.

   (b) A disparidade dos pontos linhas (círculos brancos na Figura (4.3)), é estimada em dois passos: primeiro é calculada média dos vizinhos esquerdo e direito (quadrados na Figura (4.3)), e depois a disparidade do ponto linha recebe o índice de menor MAD do 3 pontos, a média e os dois vizinhos esquerdo e direito.

(c) A disparidade dos pontos colunas (círculos brancos na Figura (4.3)), é estimada em dois passos: primeiro é calculada média dos vizinhos superior e inferior (quadrados na Figura (4.3)), e depois a disparidade do ponto coluna recebe o índice de menor MAD dos 3 pontos, a média e os dois vizinhos superior e inferior.

9. Repetir os passos 6, 7 e 8 até que o MDD tenha a resolução de $720x576$.

CAPÍTULO

5

Álgebra de Clifford

A álgebra de Clifford é um sistema adaptado à geometria ortogonal de um espaço arbitrário, que de acordo com sua assinatura pode generalizar diversos espaços. A assinatura é representada por $Cl_{p,q} = Cl_n$ onde:

p: é igual à quantidade de vetores de base que elevados ao quadrado são iguais a 1;

q: é igual à quantidade de vetores de base que elevados ao quadrado são iguais a -1;

$n = p + q$: é a soma da quantidade de p e q, e define a dimensão da álgebra de Clifford.

A álgebra de Clifford isomorfa ao espaço Euclidiano 3D de assinatura Cl_3 é composta por: um escalar 1, 3 vetores e_1, e_2, e_3, 3 bivetores e_{12}, e_{13}, e_{23}, e o trivetor ou pseudoescalar e_{123}. Sob esses objetos atuam entre outros operadores a soma geométrica, o produto geométrico, o produto interior e o produto exterior que serão detalhados a seguir.

5.1 Soma Geométrica

A soma geométrica só pode ser efetuada para objetos iguais (exemplo, $2 * e_1 + e_1 = 3 * e_1$), a soma de objetos diferentes como um vetor e um bivetor não pode ser unificada em um único termo (exemplo, $2 * e_1 + e_{12} = 2 * e_1 + e_{12}$). A soma geométrica é uma operação que respeita as propriedades da associatividade (5.1), comutatividade (5.2) e que admite o elemento nulo (5.3) (Hestenes, 2003; Dorst e Mann, 2002; Mann e Dorst, 2002).

$$x + (y + z) = (x + y) + z \tag{5.1}$$

$$x + y = y + x \tag{5.2}$$

$$x + 0 = x \tag{5.3}$$

Como a soma de objetos diferentes não pode resultar em um único termo, nós a representamos por um objeto híbrido chamado de multivetor M, que na álgebra de Clifford de assinatura Cl_3 é descrito na Equação (5.4). A soma dos multivetores é definida da mesmo forma que a soma geométrica onde são somados os termos respectivos de cada multivetor.

$$M = a_0 * 1 + a_1 * e_1 + a_2 * e_2 + a_3 * e_3 + a_4 * e_{12} + a_5 * e_{23} + a_6 * e_{31} + a_7 * e_{123} \quad (5.4)$$

5.2 Produto Geométrico

O produto geométrico, denotado por " $*$ ", entre vetores é dado pela Equação (5.5), e composto por uma parte simétrica (produto interior) e outra anti-simétrica (produto exterior). O produto geométrico apresenta as seguintes propriedades:

1. é uma operação não-comutativa $xy \neq yx$ onde x, y e z são vetores,
2. é uma operação associativa $x(yz) = (xy)z$,
3. tem a propriedade de inversa onde o vetor x possui um vetor inverso x^{-1} tal que $x * x^{-1} = 1$,
4. tem a propriedade distributiva sobre adição $x(y + z) = xy + xz$ e $(x + y)z = xz + yz$.

O produto geométrico na geometria euclidiana ortogonal é definido por duas relações geométricas, a relação (5.6) é referente ao quadrado do vetor $e_i^2 = 1$ e a relação (5.7) é referente a anti-comutatividade do produto geométrico (Hestenes, 2003; Dorst e Mann, 2002; Mann e Dorst, 2002). A Tabela (5.1) apresenta a definição do produto geométrico entre os elementos de base da álgebra de Clifford para assinatura Cl_3 ou $Cl_{3,0}$.

$$a * b = a.b + a \wedge b \quad (5.5)$$

$$e_i * e_i = 1 \quad (5.6)$$

$$e_i * e_j + e_j * e_i = 0 \quad (5.7)$$

5.3 Produto Interior

Produto interior ou também conhecido como produto de Gibbs-Heaviside, denotado por ".", quando aplicado sobre dois vetores **a** e **b** pode ser visto como a parte simétrica do produto geométrico, tem a propriedade comutativa e mede a perpendicularidade entre vetores, é dado

Cl_3	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
1	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
e_1	e_1	1	e_{12}	$-e_{31}$	e_2	$-e_3$	e_{123}	e_{23}
e_2	e_2	$-e_{12}$	1	e_{23}	$-e_1$	e_{123}	e_3	e_{31}
e_3	e_3	e_{31}	$-e_{23}$	1	e_{123}	e_1	$-e_2$	e_{12}
e_{12}	e_{12}	$-e_2$	e_1	e_{123}	-1	e_{23}	$-e_{31}$	e_3
e_{31}	e_{31}	e_3	e_{123}	$-e_1$	$-e_{23}$	-1	e_{12}	e_2
e_{23}	e_{23}	e_{123}	$-e_3$	e_2	e_{31}	$-e_{12}$	-1	e_1
e_{123}	e_{123}	e_{23}	e_{31}	e_{12}	$-e_3$	$-e_2$	$-e_1$	-1

Tabela 5.1: Produto geométrico.

pela Equação (5.8) (Hestenes, 2003; Dorst e Mann, 2002; Mann e Dorst, 2002). A Tabela (5.2) apresenta a definição do produto interior entre os elementos de base da álgebra de Clifford para assinatura Cl_3. As regras que definem o produto interior para elementos arbitrários podem ser encontradas em (Dorst e Mann, 2002; Suter, 2003).

$$a.b = 1/2(a * b + b * a) = b.a \tag{5.8}$$

Cl_3	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
1	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
e_1	0	1	0	0	e_2	$-e_3$	0	e_{23}
e_2	0	0	1	0	$-e_1$	0	e_3	e_{31}
e_3	0	0	0	1	0	e_1	$-e_2$	e_{12}
e_{12}	0	0	0	0	-1	0	0	$-e_3$
e_{31}	0	0	0	0	0	-1	0	$-e_2$
e_{23}	0	0	0	0	0	0	-1	$-e_1$
e_{123}	0	0	0	0	0	0	0	-1

Tabela 5.2: Produto interior.

5.4 Produto Exterior

O produto exterior ou também conhecido como produto de Grassmann, denotado por " $\wedge$ ", quando aplicado sobre dois vetores **a** e **b** representa a parte anti-simétrica do produto geométrico como mostra a Equação (5.9) (Hestenes, 2003; Dorst e Mann, 2002; Mann e Dorst, 2002). A Tabela (5.3) apresenta a definição do produto exterior para os elementos da álgebra de Clifford de assinatura Cl_3. As regras que definem o produto exterior para elementos arbitrários podem ser encontradas em (Dorst e Mann, 2002; Suter, 2003).

$$a \wedge b = 1/2(a * b - b * a) = -b \wedge a \tag{5.9}$$

Cl_3	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
1	1	e_1	e_2	e_3	e_{12}	e_{31}	e_{23}	e_{123}
e_1	e_1	0	e_{12}	$-e_{31}$	0	0	e_{123}	0
e_2	e_2	$-e_{12}$	0	e_{23}	0	e_{123}	0	0
e_3	e_3	e_{31}	$-e_{23}$	0	e_{123}	0	0	0
e_{12}	e_{12}	0	0	e_{123}	0	0	0	0
e_{31}	e_{31}	0	e_{123}	0	0	0	0	0
e_{23}	e_{23}	e_{123}	0	0	0	0	0	0
e_{123}	e_{123}	0	0	0	0	0	0	0

Tabela 5.3: Produto exterior.

5.5 Mapeamento das cores

Com o desenvolvimento do trabalho foram realizados três mapeamentos um para o modelo RGB e um para o modelo YUV nos vetores de base da álgebra (e_1, e_2, e_3), como mostram a Equação (5.10) e a Equação (5.11), e outro para o modelo YUV onde a luminância Y e mapeada no escalar (1) e as crominâncias UV nos vetores (e_1, e_2) como mostra a Equação (5.12), cujos subscritos x e y são as coordenadas do pixel $P_{x,y}$.

$$P_{x,y} = R_{x,y} * e_1 + G_{x,y} * e_2 + B_{x,y} * e_3 \tag{5.10}$$

$$P_{x,y} = Y_{x,y} * e_1 + U_{x,y} * e_2 + V_{x,y} * e_3 \tag{5.11}$$

$$P_{x,y} = Y_{x,y} * 1 + U_{x,y} * e_1 + V_{x,y} * e_2 \tag{5.12}$$

5.6 Convolução em Clifford

A convolução em Clifford (Mishra et al., 2008; Ebling e Scheuermann, 2003), análoga a convolução no domínio Real, é realiza através do produto geométrico entre o filtro (h) e a imagem (P), ambos definidos no domínio de Clifford como mostra a Equação (5.13).

$$P(m,n) = \sum_{x=-X}^{X} \sum_{y=-Y}^{Y} h(x,y) * P(m-x, n-y) \tag{5.13}$$

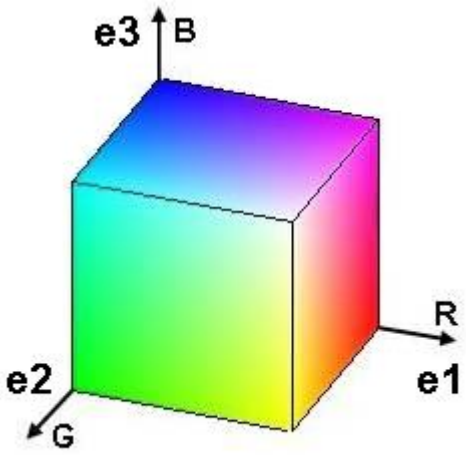


Figura 5.1: Mapeamento dos componentes RGB nos vetores de base (e_1, e_2, e_3).

5.7 Norma de Clifford

A norma dos multivetores é calculada de forma similar ao valor absoluto aplicado sobre o conjunto dos números complexos, ou seja, assumindo que A seja um multivetor então a norma de $|A|$, é dada pela Equação (5.14), e resulta em um único valor positivo que representa a grandeza do multivetor. O reverso $\tilde{A}$ é obtido através da inversão dos versores i, j e k de cada termo de A, onde o bivetor e_{ij} tem como reverso o bivetor $-e_{ji}$, neste caso o bivetor fica negativo porque a cada permutação devemos mudar o sinal do bivetor.

Exemplo: seja $A = 1 + e_1 + e_2 + e_3 + e_{12} + e_{23} + e_{31} + e_{123}$,

seu reverso é $\tilde{A} = 1 + e_1 + e_2 + e_3 - e_{21} - e_{32} - e_{13} + e_{321}$ (Suter, 2003).

$$|A| = \sqrt{A\tilde{A}} \tag{5.14}$$

5.8 Mean Absolute Difference em Clifford

Como o Mean Absolute Difference (MAD) é definido sobre o domínio real, temos que definir um MAD em Clifford (que chamamos de MADC), que aplicado sobre um multivetor retorne um valor real positivo que represente sua magnitude. Utilizando a norma dos multivetores e a Equação (2.1), se define o MADC como mostra a Equação (5.15), onde $d(m, n)$ armazena um multivetor e não mais um numero real.

$$mdc(dx, dy) = min|d(m, n)| \tag{5.15}$$

CAPÍTULO

6

Extensão do Quad-Tree

Utilizando a abordagem proposta por Chiari (Chiari et al., 1997) e a álgebra de Clifford, o algoritmo Quad-tree pode ser estendido para trabalhar com imagens coloridas. A extensão em Clifford segue a estrutura presente na Figura (6.1) com as modificações mostradas em vermelho, onde foram realizadas as alterações necessárias para utilizar a álgebra de Clifford.

Seguindo a descrição do Quad-tree foram feitas as seguintes alterações:

1. para o passo 1, a imagem colorida é definida em Clifford (Ebling e Scheuermann, 2003; Mishra et al., 2008; Mishra e Wilson, 2006) como descrito anteriormente para o modelo RGB utilizando a Equação (5.10), e para o modelo YUV as Equações (5.11) e (5.12);

2. no passo 2 o filtro da Tabela (4.1) e Figura (4.2) é codificado no escalar 1 do domínio de Clifford e se utiliza a convolução da Equação (5.13), onde a multiplicação vira produto geométrico e as variáveis não são mais reais e sim números de Clifford;

3. os passos 3 e 4 não são alterados;

4. no passo 5 é utilizado o MADC, Equação (5.15), para determinar os pixels correspondentes;

5. os passos 6, 7, 8 e 9 seguem da mesma forma sem alteração.

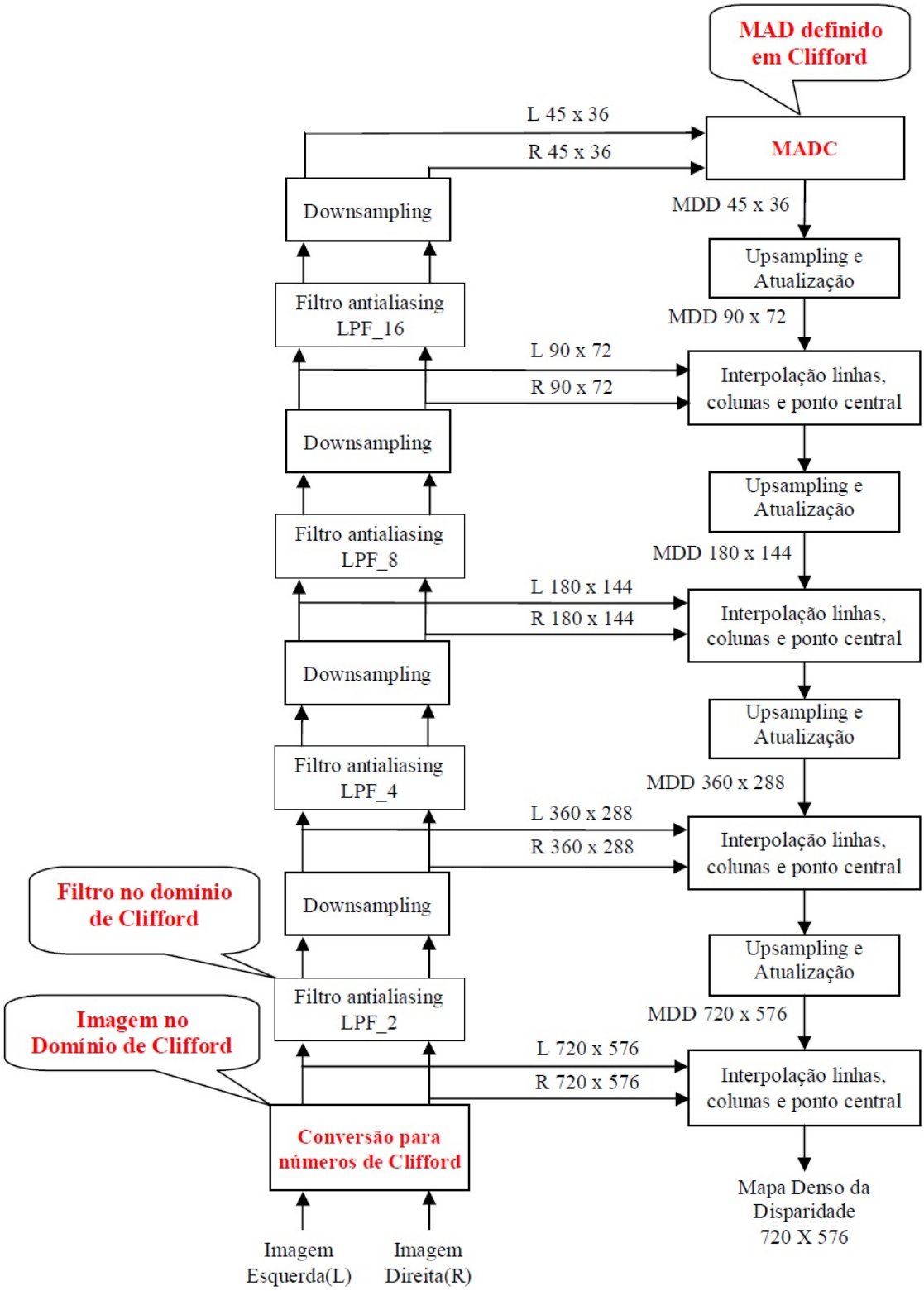


Figura 6.1: Quad-Tree estendido no domínio de Clifford.

CAPÍTULO

7

Resultados e Discussão dos Resultados

Nesta seção, apresentamos os resultados dos experimentos realizados com a luminância no domínio Real e os modelos RGB e YUV no domínio de Clifford. As imagens estereoscópicas utilizadas nos experimentos são de $720x576$ pixels, e apresentam desde estruturas mais simples com apenas um objeto em cena, a estruturas mais complexas com vários objetos a distâncias diferentes na mesma imagem. A Figura (7.1) é o exemplo de par de imagens estereoscópicas complexas utilizadas nos experimentos.

Os experimentos foram realizados com três mapeamentos das cores no domínio de Clifford, no primeiro os valores RGB da imagem foram mapeados nos vetores ($R * e_1$, $G * e_2$, $B * e_3$), no segundo os valores YUV também foram mapeados no vetores ($Y * e_1$, $U * e_2$, $V * e_3$), e no terceiro a luminância Y é mapeada no escalar ($Y * 1$) e as crominâncias UV nos vetores ($U * e_1$, $V * e_2$), que será referenciado no texto como YUV2 para diferenciar os dois mapeamentos utilizados no modelo YUV.

Os resultados foram obtidos utilizando-se três programas implementados na linguagem JAVA, no primeiro é calculado e criado o mapa denso da disparidade utilizando a biblioteca de **Geometric AlgeBra Learning Environment (GABLE)** (Mann et al., 2001), que possibilita o cômputo dos principais operadores das álgebras geométricas de Cliffod de assinatura $Cl_{p,q}$, para o caso específico das álgebras onde $p + q = 3$; o segundo programa realiza a reconstrução da imagem direita utilizando o mapa denso da disparidade e a imagem esquerda; e o terceiro programa calcula o erro quadrático médio entre a imagem direita e a reconstruída.

O mapa denso da disparidade gerado foi representado utilizando tons de cinza onde os objetos próximos ao observador possuem tons de cinza mais claros e os distantes os tons de

(a) Imagem esquerda. (b) Imagem direita.

Figura 7.1: Par estereoscópico do carrossel.

cinza mais escuros. Foram realizadas reconstruções utilizando o mapa gerado, e através destas reconstruções, calculada a relação sinal/ruído SNR utilizando as Equações (7.1) e (7.2), entre a imagem reconstruída e a original, a fim de verificar a qualidade da imagem reconstruída o que pode ser observado através da análise visual dos resultados.

$$diferenca_{(x,y)} = (R_{O(x,y)} - R_{R(x,y)})^2 + (G_{O(x,y)} - G_{R(x,y)})^2 + (B_{O(x,y)} - B_{R(x,y)})^2 \quad (7.1)$$

$$eqm = \frac{1}{height * width * 3} \sum_{x,y}^{x=height,y=width} \sqrt{diferenca(x,y)} \quad (7.2)$$

em que:

- $diferenca$ = variável que armazena a diferença de dois pixels RGB.
- eqm = variável que armazena o resultado do erro quadrático médio.
- $R_{O(x,y)}$ = valor da componente $R_{(x,y)}$ da imagem original.
- $R_{R(x,y)}$ = valor da componente $R_{(x,y)}$ da imagem reconstruída.
- $G_{O(x,y)}$ = valor da componente $G_{(x,y)}$ da imagem original.
- $G_{R(x,y)}$ = valor da componente $G_{(x,y)}$ da imagem reconstruída.
- $B_{O(x,y)}$ = valor da componente $B_{(x,y)}$ da imagem original.
- $B_{R(x,y)}$ = valor da componente $B_{(x,y)}$ da imagem reconstruída.

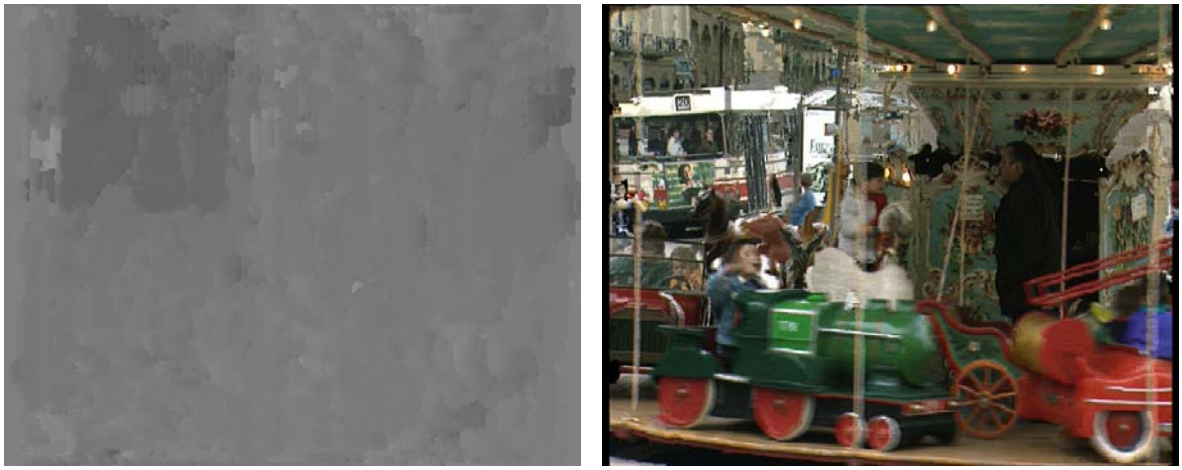
As Figuras (7.2), (7.3), (7.4) e (7.5), apresentam os resultados obtidos para imagens do Carrossel (Figura(7.1)), utilizando luminância e a metodologia proposta com álgebra de Clifford

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.2: Resultados sobre a Luminância das imagens do carrossel

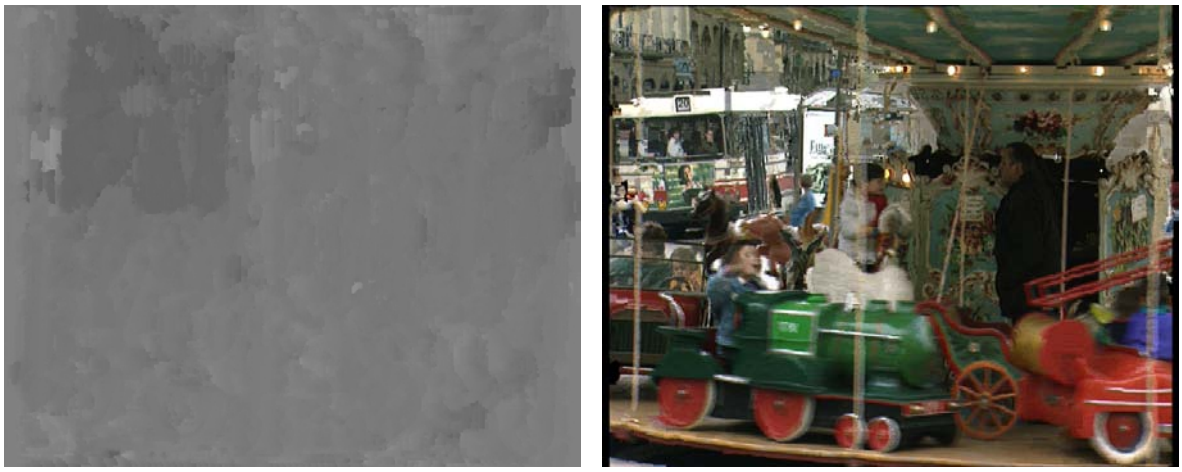
(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.3: Resultados de Clifford com RGB das imagens do carrossel



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.4: Resultados de Clifford com YUV das imagens do carrossel



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.5: Resultados de Clifford com YUV2 das imagens do carrossel

Imagem	Luminância	RGB	YUV	YUV2
Carrossel 0	27.9873	27.9597	27.9003	27.9013
Carrossel 1	28.0163	27.9557	27.8934	27.8941
Carrossel 2	27.9567	27.9521	27.8980	27.8976
Carrossel 3	27.9550	27.8996	27.8514	27.8481
Carrossel 4	27.9881	27.9531	27.8976	27.8897
Carrossel 5	27.9879	27.9919	27.9211	27.9141
Carrossel 6	28.0283	28.0165	27.9608	27.9591
Carrossel 7	28.1246	28.1201	28.0789	28.0797
Carrossel 8	28.1647	28.1366	28.0746	28.0718
Carrossel 9	28.1283	28.1095	28.0474	28.0489

Tabela 7.1: Resultados do SNR obtidos das imagens do carrossel.

para os modelos RGB e YUV, respectivamente. Em uma análise visual dos resultados obtidos nas imagens do Carrossel, podemos encontrar em regiões da imagem reconstruída (Figura (7.3(b)), Figura (7.4(b)) e Figura (7.5(b))) algumas pessoas que estão presente dentro do ônibus e o número que identifica a rota do ônibus "20" que são visíveis em Clifford e não são visíveis na imagem reconstruída utilizando apenas luminância (Figura (7.2(b))). A reconstrução nas áreas oclusas obviamente esta errada já que não se pode resolver o problema da correspondência, pois o pixel esta presente apenas em uma das imagens direita ou esquerda, então o método aproxima o pixel em questão com o pixel que obtiver o menor MAD. Isto é observado na senhora que aparece do lado esquerdo na imagem direita (Figura (7.1(b))), que não aparece na imagem reconstruída. Na Tabela (7.1), são apresentados alguns dos valores do SNR obtidos utilizando luminância e Clifford com RGB e YUV para o conjunto de imagens do Carrossel, onde se observa que o valor do SNR foi melhor para os resultados obtidos pela luminância, entretanto com a analise visual das imagens é possível identificar que os resultados em Clifford são melhores, o que destaca a importância da análise subjetiva dos resultados.

(a) Imagem esquerda. (b) Imagem direita.

Figura 7.6: Par estereoscópico do trem.

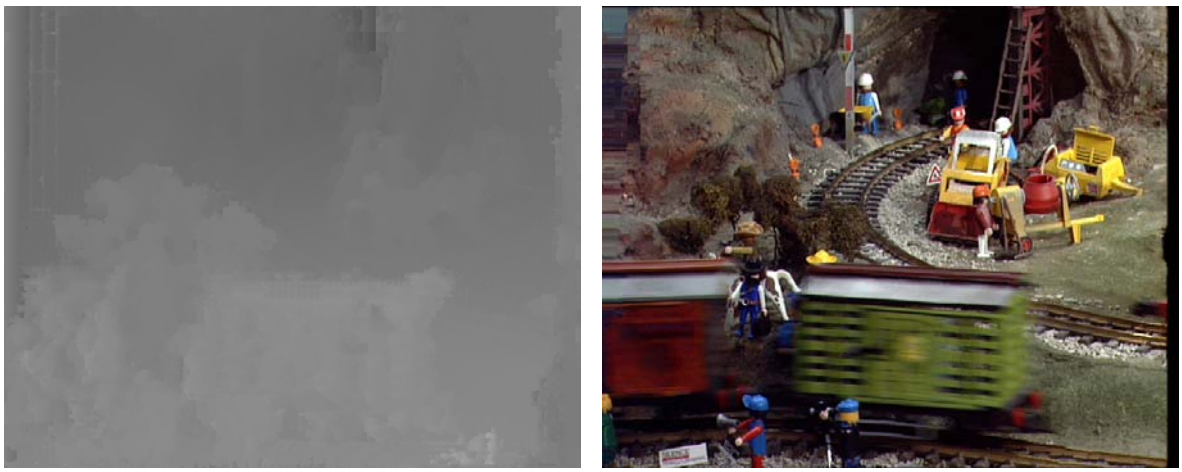
As Figuras (7.7), (7.8), (7.9), (7.10) e (7.11), apresentam os resultados obtidos para imagens do Trem (Figura (7.6)), utilizando luminância e a metodologia proposta com álgebra de Clifford para os modelos RGB e YUV, respectivamente. Algumas das sequências de imagens usadas apresentam variação na iluminação entre o par estéreo, o que fez com que o matching fosse errôneo em algumas regiões da imagem. Assim pode-se observar, nos mapas presentes nas Figuras (7.8(a)) e (7.10(a)), que a diferença de iluminação causou o erro no matching em algumas regiões. No mapa denso da disparidade tanto em RGB quanto em YUV da Figura (7.8(a)) e Figura (7.10(a)), respectivamente, é possível ver uma região mais escura na frente do primeiro trenzinho, o que não acontece no mapa da Figura (7.9(a)) e Figura (7.11(a)). Na Tabela (7.2), são apresentados alguns dos valores do SNR obtidos utilizando luminância e Clifford com RGB e YUV para o conjunto de imagens do Trem, onde é possível verificar que das 10 imagens

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.7: Resultados sobre a Luminância das imagens do trem.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.8: Resultados de Clifford com RGB das imagens do trem para cena 1.

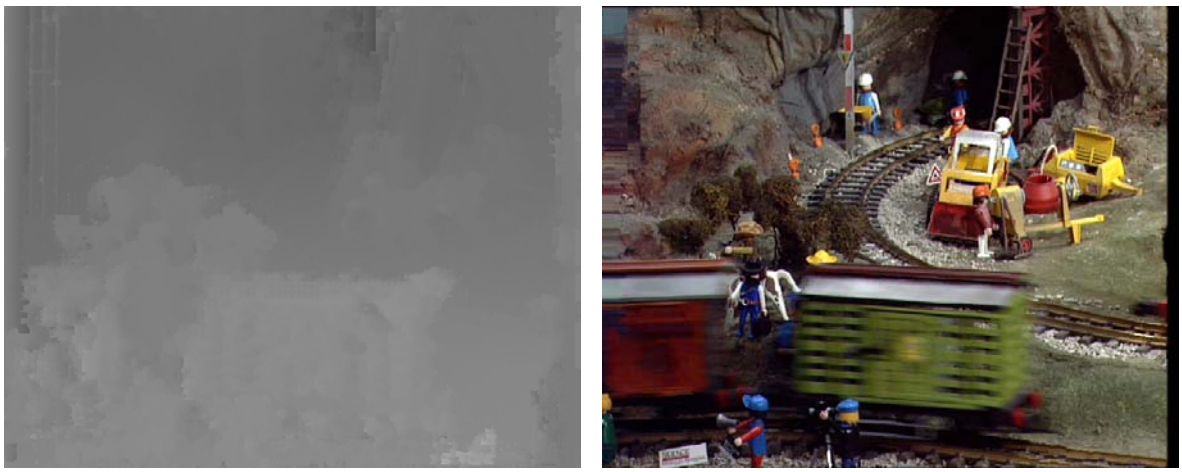


(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.9: Resultados de Clifford com RGB das imagens do trem para cena 2.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.10: Resultados de Clifford com YUV das imagens do trem para cena 1.

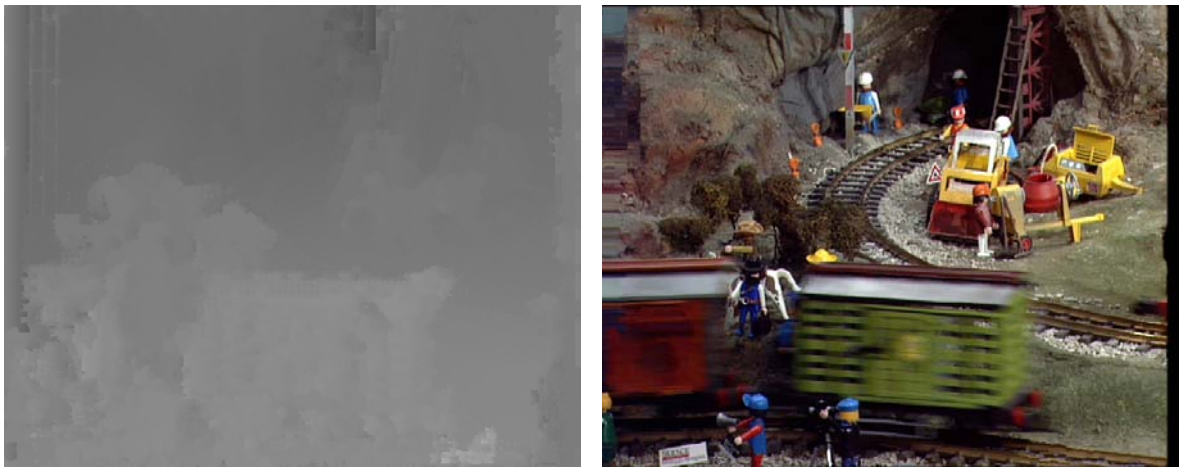


(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.11: Resultados de Clifford com YUV das imagens do trem para cena 2.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.12: Resultados de Clifford com YUV2 das imagens do trem para cena 1.



(a) Mapa denso da disparidade.

(b) Imagem reconstruída.

Figura 7.13: Resultados de Clifford com YUV2 das imagens do trem para cena 2.

Imagem	Luminância	RGB	YUV	YUV2
Trem 0	30.0362	29.8586	29.8461	29.8458
Trem 1	29.9589	29.8081	29.7863	29.7858
Trem 2	29.8870	29.7714	29.7754	29.7750
Trem 3	29.8667	29.8166	29.7946	29.7944
Trem 4	29.8283	29.7814	29.7518	29.7519
Trem 5	29.6283	29.7817	29.7227	29.7228
Trem 6	29.6052	29.7392	29.7156	29.7155
Trem 7	29.5867	29.7436	29.7100	29.7093
Trem 8	29.4674	29.6546	29.6443	29.6449
Trem 9	29.5758	29.7123	29.6956	29.6957

Tabela 7.2: Resultados do SNR obtidos das imagens do trem.

testadas, as 5 primeiras imagens onde existe uma variação na iluminação entre o par estéreo a luminância obteve melhores valores de SNR, já nas outras 5 imagens onde não existe essa variação Clifford obteve melhores valores de SNR.

(a) Imagem esquerda. (b) Imagem direita.

Figura 7.14: Par estereoscópico do rinoceronte.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.15: Resultados sobre a Luminância das imagens do rinoceronte.

As Figuras (7.15), (7.16), (7.17) e (7.18), apresentam os resultados obtidos para imagens do Rinoceronte (Figura (7.14)), utilizando luminância e a metodologia proposta com álgebra de Clifford para os modelos RGB e YUV, respectivamente. As imagens do rinoceronte apresentam estruturas simples com apenas um objeto em cena, onde é possível verificar que os resultados do SNR obtidos utilizando Clifford são melhores aos obtidos utilizando a luminância como pode ser observado na Tabela (7.3). Nos mapas gerados em Clifford, as regiões homogêneas do fundo da imagem apresentam regiões claras e escuras que estão erradas, as regiões claras ocorrem quando o MAD é menor para os últimos índices da área de busca, que indicam um ponto mais próximos, as regiões escuras ocorrem quando o MAD é menor para os primeiros

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.16: Resultados de Clifford com RGB das imagens do rinoceronte.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.17: Resultados de Clifford com YUV das imagens do rinoceronte.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.18: Resultados de Clifford com YUV2 das imagens do rinoceronte.

Imagem	Luminância	RGB	YUV	YUV2
Rinoceronte 0	29.6636	29.6728	29.6854	29.6854
Rinoceronte 1	29.6462	29.6655	29.6504	29.6504
Rinoceronte 2	29.6830	29.6881	29.6865	29.6865
Rinoceronte 3	29.6977	29.7314	29.7114	29.7114
Rinoceronte 4	29.7070	29.7451	29.7253	29.7253
Rinoceronte 5	29.7330	29.7424	29.7345	29.7345
Rinoceronte 6	29.7412	29.7403	29.7436	29.7437
Rinoceronte 7	29.7523	29.7688	29.7607	29.7607
Rinoceronte 8	29.7142	29.7251	29.7250	29.7249
Rinoceronte 9	29.7119	29.7304	29.7204	29.7204

Tabela 7.3: Resultados do SNR obtidos das imagens do rinoceronte.

índices da área de busca, que indicam pontos mais distantes. Na analise visual dos resultados pode-se identificar que os resultados utilizando a luminância são melhores para as imagens do rinoceronte, mesmo que os valores de SNR apontem que Clifford teria melhores resultados.

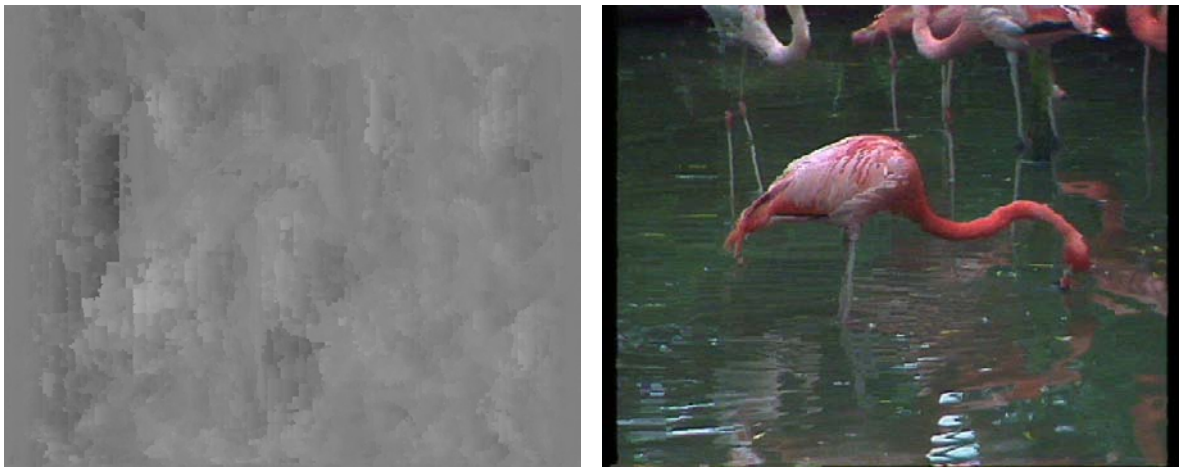
(a) Imagem esquerda. (b) Imagem direita.

Figura 7.19: Par estereoscópico do flamingo.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.20: Resultados sobre a Luminância das imagens do flamingo.

As Figuras (7.20), (7.21), (7.22) e (7.23), apresentam os resultados obtidos para imagens do Flamingo (Figura (7.19)), utilizando luminância e a metodologia proposta com álgebra de Clifford para os modelos RGB e YUV, respectivamente. Na Tabela (7.4), são apresentados alguns dos valores do SNR obtidos utilizando luminância e Clifford com RGB e YUV para o conjunto de imagens do Flamingo, onde é possível verificar que a luminância obteve melhores valores de SNR em 10 das 10 imagens testadas. As reconstruções obtidas nas imagens do Flamingo foram semelhantes nos resultados obtidos tanto com a luminância quanto em Clifford, os mapas gerados em Clifford apresentaram alguns erros com pontos pretos na parte da frente da imagem, que ocorrem em algumas áreas homogêneas da imagem.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.21: Resultados de Clifford com RGB das imagens do flamingo.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

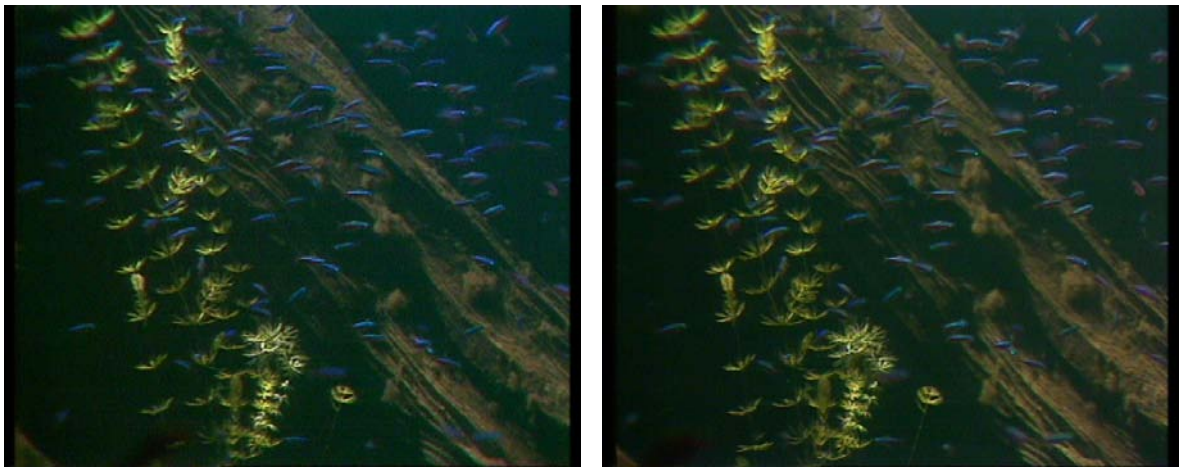
Figura 7.22: Resultados de Clifford com YUV das imagens do flamingo.

(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.23: Resultados de Clifford com YUV2 das imagens do flamingo.

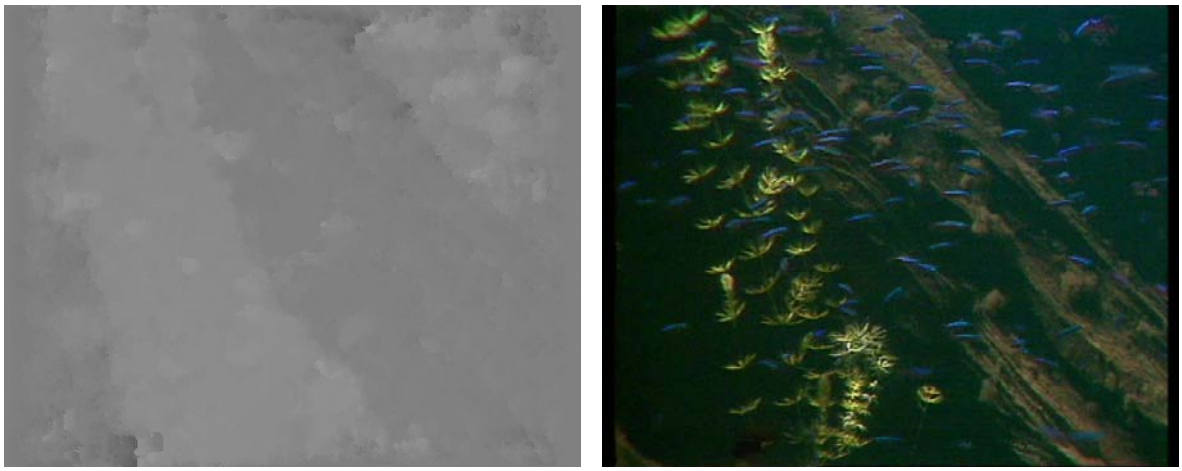
Imagem	Luminância	RGB	YUV	YUV2
Flamingo 0	28.5530	28.4936	28.4681	28.4681
Flamingo 1	28.5051	28.4782	28.4472	28.4472
Flamingo 2	28.4961	28.4885	28.4653	28.4653
Flamingo 3	28.5008	28.4395	28.4465	28.4464
Flamingo 4	28.5513	28.4994	28.5094	28.5094
Flamingo 5	28.5556	28.5311	28.5087	28.5087
Flamingo 6	28.5798	28.5530	28.5340	28.5340
Flamingo 7	28.5920	28.5574	28.5233	28.5233
Flamingo 8	28.6986	28.6342	28.6126	28.6126
Flamingo 9	28.6255	28.6052	28.5822	28.5822

Tabela 7.4: Resultados do SNR obtidos das imagens do flamingo.



(a) Imagem esquerda. (b) Imagem direita.

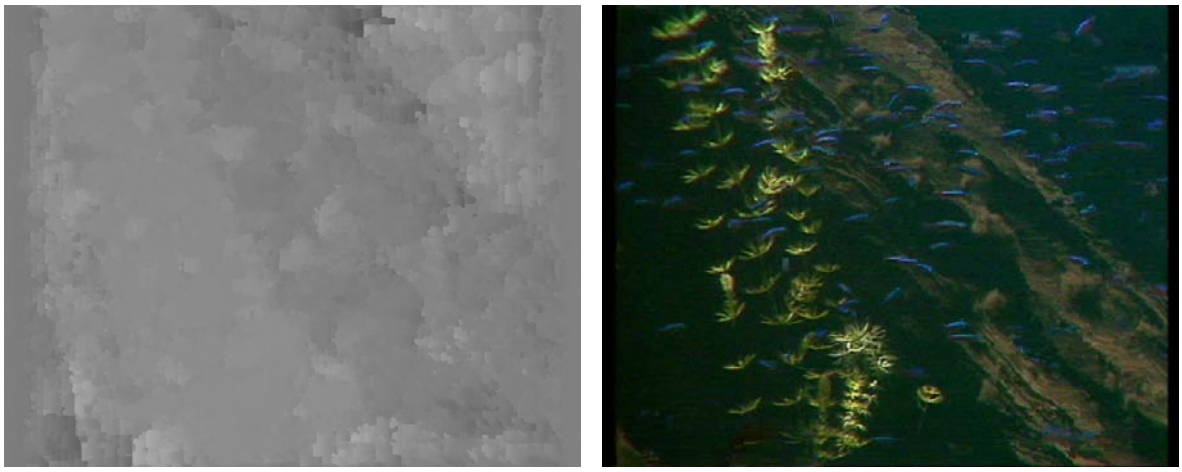
Figura 7.24: Par estereoscópico dos peixes.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

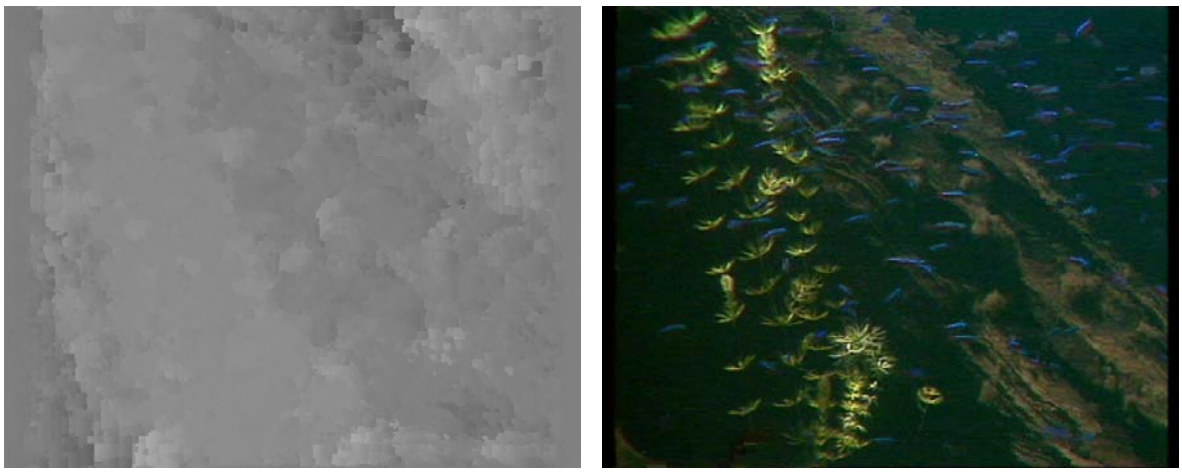
Figura 7.25: Resultados sobre a Luminância das imagens dos peixes.

As Figuras (7.25), (7.26), (7.27) e (7.28), apresentam os resultados obtidos para imagens dos Peixes (Figura (7.24)), utilizando luminância e a metodologia proposta com álgebra de Clifford para os modelos RGB e YUV, respectivamente. As imagens dos peixes obtiveram bons resultados para ambos os métodos, os resultados em Clifford do mapa denso da disparidade apresentam uma maior sensibilidade as diferentes distancias presentes na imagem, tendo assim, uma melhor representatividade da cena, já o mapa denso da disparidade da luminância é mais homogêneo, o que demostra menor sensibilidade as diferentes distâncias persentes na cena. Os valores de SNR também foram muito próximos como pode ser visto na Tabela 7.5, onde é possível identificar que 5 imagens obtiver melhores valores de SNR para Clifford e as outras 5 imagens para a luminância.



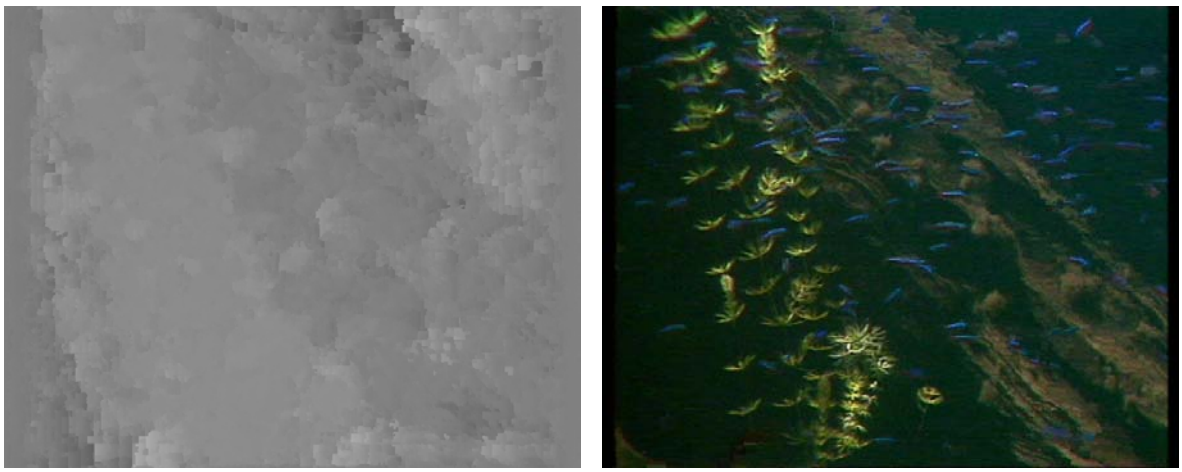
(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.26: Resultados de Clifford com RGB das imagens dos peixes.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.27: Resultados de Clifford com YUV das imagens dos peixes.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

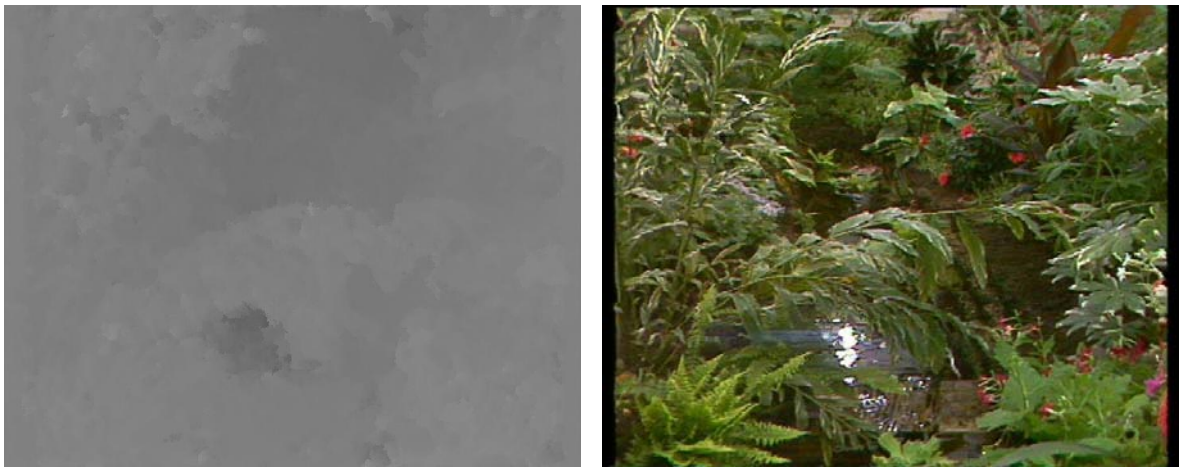
Figura 7.28: Resultados de Clifford com YUV2 das imagens dos peixes.

Imagem	Luminância	RGB	YUV	YUV2
Peixes 0	25.1679	25.1904	25.1395	25.1395
Peixes 1	25.1439	25.1761	25.1370	25.1370
Peixes 2	25.1313	25.1508	25.1023	25.1023
Peixes 3	25.1376	25.1363	25.1041	25.1041
Peixes 4	25.1379	25.1278	25.0791	25.0791
Peixes 5	25.1146	25.0930	25.0304	25.0304
Peixes 6	25.1333	25.1464	25.1079	25.1079
Peixes 7	25.0679	25.0597	25.0357	25.0357
Peixes 8	25.1050	25.1313	25.0806	25.0806
Peixes 9	25.1065	25.1009	25.0686	25.0686

Tabela 7.5: Resultados do SNR obtidos das imagens dos peixes.

(a) Imagem esquerda. (b) Imagem direita.

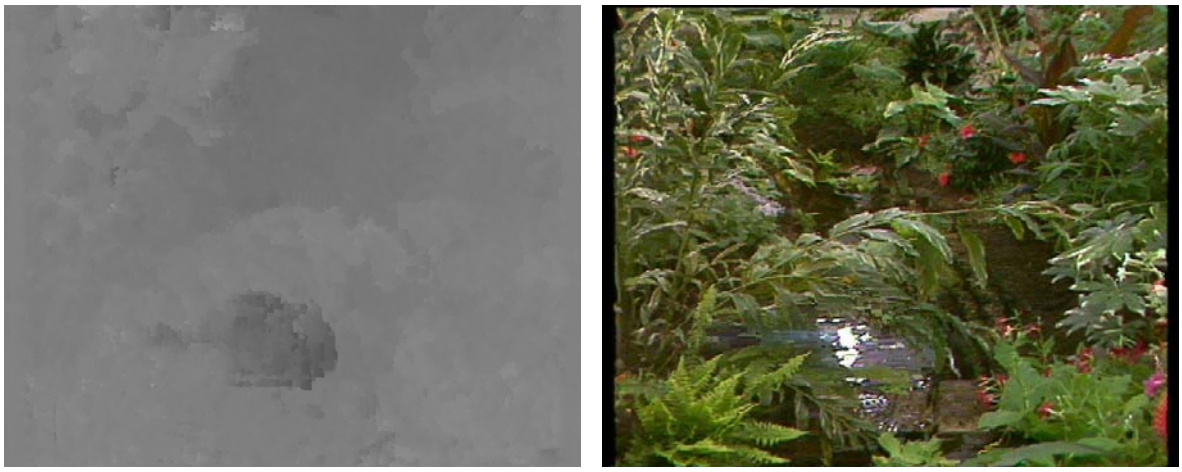
Figura 7.29: Par estereoscópico do riacho.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

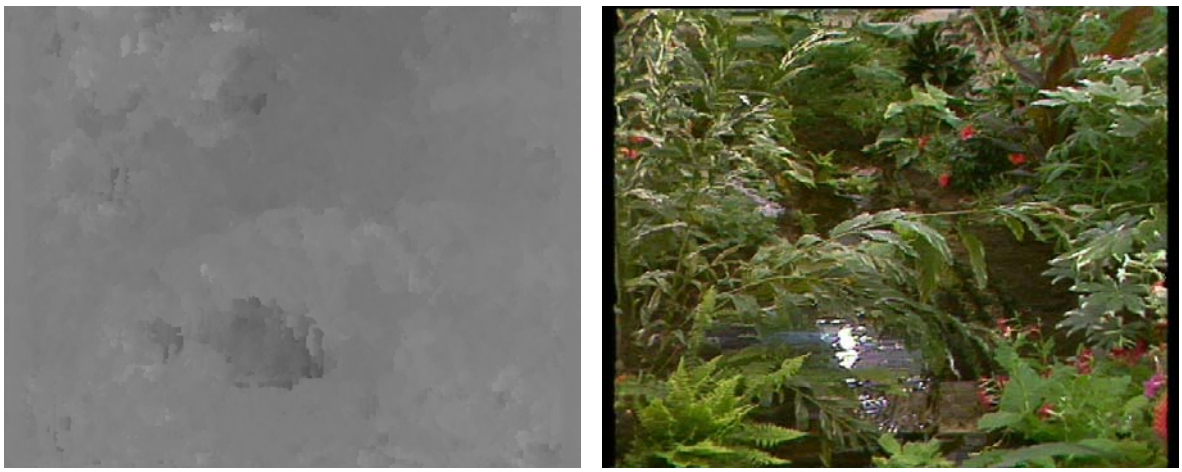
Figura 7.30: Resultados sobre a Luminância das imagens do riacho.

Os resultados obtidos para as sequência de imagens do Riacho (Figura (7.29)), são semelhantes entre os métodos utilizados como mostram as Figuras (7.30), (7.31), (7.32) e (7.33). Os mapas gerados em Clifford são menos homogêneo que o mapa gerado pela luminância, para as imagens do Riacho. Os resultados em Clifford mostram uma melhor sensibilidade as diferentes profundidades presentes a cena, pode-se notar que a uma área escura na parte da frente da imagem do riacho em todos os mapas gerados (Figuras (7.30), (7.31), (7.32) e (7.33)), entretanto nos mapas gerados em Clifford é possível identificar uma área maior se comparado ao mapa gerado pela luminância. Nesses casos a informação de cor presente nos modelos utilizados RGB e YUV, obteve vantagem sobre a luminância, as informações de cor presentes na imagem colaboraram para a escolha do pixel correspondente dessas regiões. Na Tabela 7.6, são apresentados alguns dos valores do SNR obtidos utilizando luminância e Clifford com RGB e YUV



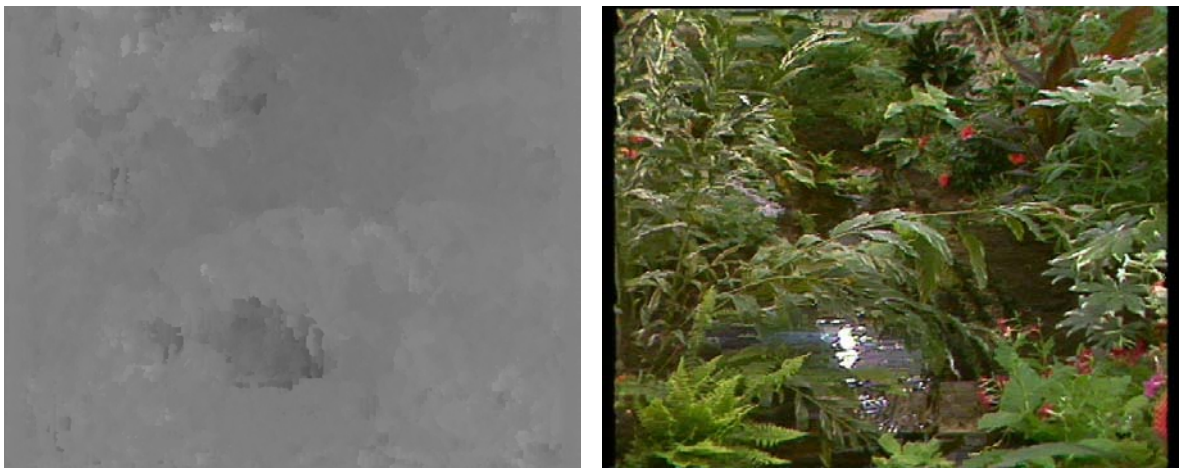
(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.31: Resultados de Clifford com RGB das imagens do riacho.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.32: Resultados de Clifford com YUV das imagens do riacho.



(a) Mapa denso da disparidade. (b) Imagem reconstruída.

Figura 7.33: Resultados de Clifford com YUV2 das imagens do riacho.

Imagem	Luminância	RGB	YUV	YUV2
Riacho 0	26.8301	26.7645	26.7560	26.7559
Riacho 1	26.8455	26.7959	26.7687	26.7687
Riacho 2	26.8619	26.8088	26.8065	26.8065
Riacho 3	26.7874	26.7335	26.7189	26.7187
Riacho 4	26.8293	26.7633	26.7567	26.7564
Riacho 5	26.8094	26.7633	26.7334	26.7331
Riacho 6	26.7851	26.7281	26.7223	26.7225
Riacho 7	26.7626	26.7227	26.6825	26.6825
Riacho 8	26.7968	26.7409	26.7279	26.7279
Riacho 9	26.8121	26.7668	26.7457	26.7451

Tabela 7.6: Resultados do SNR obtidos das imagens do riacho.

para o conjunto de imagens do Riacho, onde se observa que os valores de SNR apontam que a luminância teria melhores resultados, o que não é confirmado pela análise visual do resultados.

A extensão do Quad-tree aqui apresentada para estimação do mapa denso da disparidade utilizando imagens coloridas no domínio de Clifford mostrou-se eficiente para imagens mais complexas de cenas reais. Através das reconstruções obtidas nesses casos é possível visualizar a melhora nos resultados obtidos, o que indica que o mapa gerado tem uma melhor representação de profundidade da cena.

Em imagens de menor complexidade verificamos que tanto utilizando Clifford quanto luminância os resultados foram semelhantes, o que mostra que a informação de cor nesses casos é pouco relevante. Aplicando Clifford em regiões da imagem que apresentavam diferença de iluminação entre o par estereoscópico, foi possível identificar que o MAD resultou em um valor de disparidade errôneo, indicando que Clifford é sensível as variações de iluminação presente nas cenas.

Como podemos ver nos resultados obtidos dos dois mapeamentos utilizando o modelo YUV, não apresentaram diferença significativa, o que mostra que, para esse caso especifico onde o filtro esta mapeado no escalar a forma com que os valores de cor são mapeados é irrelevante.

Analisando os valores do SNR podemos perceber a importância da analise subjetiva, pois em alguns casos o SNR indicava que a luminância ou Clifford teriam melhores resultados o que não foi confirmado na analise visual.

CAPÍTULO

8

Conclusões

Extensões do Quad-tree para a estimação do mapa denso da disparidade utilizando a luminância no domínio real já foram propostas na literatura. Neste trabalho o Quad-tree foi estendido para processar imagens coloridas (RGB e YUV) codificadas no domínio da álgebra de Clifford. O mapa denso da disparidade é obtido através do cálculo da correspondência para cada pixel entre duas imagens estereoscópicas. O cálculo da correspondência é uma tarefa fundamental que apresenta problemas devido à presença de áreas oclusas, ruídos e variações na iluminação, entre outros. A álgebra de Clifford foi utilizada para mapear os valores de cor de cada pixel nos vetores de base, compondo assim um número de Clifford. No domínio de Clifford proposto, a extensão do Quad-tree gerou resultados melhores para imagens mais complexas se comparados com os obtidos sobre as imagens utilizando o domínio real sobre a luminância. Processando vários pares estereoscópicos no domínio real (luminância) e no domínio de Clifford (cores) se demonstrou experimentalmente a melhora obtida para imagens em cenas reais.

Os pixels da imagem podem ser codificados de diversas formas além da aqui utilizada, onde as componentes de cor (RGB e YUV) são mapeadas em e_1, e_2, e e_3, como alternativa podemos mapeá-los no escalar, vetor, bivetor, etc. Em raciocínio similar, a codificação do filtro somente no escalar simplifica a computação, mas pode ser codificado no escalar e replicado sobre os vetores, obtendo-se bastante redundância em cada dimensão do número de Clifford. Os experimentos realizados mostram que a ideia apresentada neste trabalho é um bom compromisso entre simplicidade e resultados obtidos. Fundamentalmente, cada dimensão dos vetores do domínio de Clifford, quando efetuada a convolução, possui uma versão filtrada tanto para cada valor de luminância Y, crominância U ou V do modelo YUV, quanto para as componentes de cor R, G e B do modelo RGB. Se o filtro fosse definido não somente no escalar, mas em todas as dimensões do domínio de Clifford teríamos a versão filtrada como antes e, adicionalmente, versões

misturadas de luminância e crominância YU, YV, UV no modelo YUV e RG, GB e RB no modelo RGB dependendo da forma em que se codifique o filtro.

Como trabalhos futuros, podem ser realizados experimentos utilizando outros modelos de cor como o HSV, outras formas de mapeamento das componentes de cor e do filtro de suavização no domínio de Clifford, assim como também pode ser utilizado na segmentação de imagens e na implementação em hardware do algoritmo.

Referências Bibliográficas

BANKS, J.; CORKE, P. Quantitative Evaluation of Matching Methods and Validity Measures for Stereo Vision. *The International Journal of Robotics Research*, v. 20, n. 7, p. 512–532, 2001.
Disponível em `http://ijr.sagepub.com/cgi/content/abstract/20/7/512`

BHAT, D.; NAYAR, S. Ordinal Measures for Image Correspondence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, v. 20, n. 4, p. 415–423, 1998.

BROWN, M. Z.; BURSCHKA, D.; HAGER, G. D. Advances in Computational Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 25, n. 8, p. 993–1008, 2003.

CHIARI, A.; CICIANI, B.; ROMERO, M. A Fault-Robust SPMD Architecture for 3D-TV Image Processing. *Proceedings of the 2nd. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, 1997.

DHOND, U.; AGGARWAL, J. Structure from Stereo-a Review. *Systems, Man and Cybernetics, IEEE Transactions on*, v. 19, n. 6, p. 1489–1510, 1989.

DORST, L.; MANN, S. Geometric Algebra: A Computational Framework for Geometrical Applications (Part 1). *IEEE Computer Graphics and Applications*, v. 22, n. 3, p. 24–31, 2002.

EBLING, J.; SCHEUERMANN, G. Clifford Convolution And Pattern Matching On Vector Fields. In: *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, Washington, DC, USA: IEEE Computer Society, p. 26, 2003.

HESTENES, D. Oersted Medal Lecture 2002: Reforming the Mathematical Language of Physics. *Am. J. Phys.*, v. 71, n. 2, p. 104–121, 2003.

HÜLLERMEIER, E.; KRUSE, R.; HOFFMANN, F. *Information Processing and Management of Uncertainty in Knowledge–Based Systems: 13th International Conference, Ipmu 2010,*

Dortmund, Germany, June 28-July 2, 2010. Proceedings. N. pt. 2 in Communications in Computer and Information Science. Springer, 2010.
Disponível em `http://books.google.com.br/books?id=QAVAbwbiszAC`

MANN, S.; DORST, L. Geometric Algebra: A Computational Framework for Geometrical Applications (Part 2). *Computer Graphics and Applications, IEEE*, v. 22, n. 4, p. 58 – 67, 2002.

MANN, S.; DORST, L.; BOUMA, T. The making of GABLE: A Geometric Algebra Learning Environment in Matlab. In: *Geometric Algebra With Applications in Science and Engineering (Ixtapa-Zihuatanejo, 1999)*, Boston, MA: Birkhäuser Boston, p. 491–511, 2001.

MISHRA, B.; WILSON, P. Color Edge Detection Hardware based on Geometric Algebra. In: *The 2nd Institution of Engineering and Technology Multimedia Conference 2006*, 2006.
Disponível em `http://eprints.ecs.soton.ac.uk/13188/`

MISHRA, B.; WILSON, P.; AL-HASHIMI, B. Advancement in Color Image Processing Using Geometric Algebra. In: *EUSIPCO*, 2008.
Disponível em `http://eprints.ecs.soton.ac.uk/16670/`

MÜHLMANN, K.; MAIER, D.; HESSER, J.; MÄNNER, R. Calculating Dense Disparity Maps from Color Stereo Images, an Efficient Implementation. *Int. J. Comput. Vision*, v. 47, n. 1-3, p. 79–88, 2002.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. *Discrete-time signal processing (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

PLATANIOTIS, K.; VENETSANOPOULOS, A. *Color Image Processing and Applications*. Digital signal processing. Springer, 2000.
Disponível em `http://books.google.com.br/books?id=vUM9-HDespYC`

SUTER, J. Geometric Algebra Primer. [*On-line*].
Disponível em `www.jaapsuter.com` (Acessado em 17/11/2010)

APÊNDICE

A

Códigos Fonte Estereoscopia em Clifford

```
import java.util.ArrayList;
import subspace.basis.BasisBlade;
import subspace.basis.Multivector;
import ij.IJ;
import ij.ImagePlus;
import ij.io.FileSaver;


public class Estereo {

  ImagePlus imagePlusRight;
  ImagePlus imagePlusLeft;

  //Icones utilizados para DownSample e UpSample
  Multivector iconeL[][][];
  Multivector iconeR[][][];

  //Filtro utilizado no Upsample e DowSample
  Multivector filterFirL[];

  //Classe de funcões para conversão de Multivector em Imagens e vice versa.
  CreateImage createImage;

  //Matriz que armazena o mapa de disparidade das imagens
  int disparidadeCol[][][]; //disparidade horizontal
```

```
int disparidadeLin[][][]; //disparidade vertical
int disparidadeColLR[][]; //disparidade horizontal
int disparidadeLinLR[][]; //disparidade vertical
int iconeCol[][][];
int iconeLin[][][];

/**
 * Armazena o número de decimações a serem realizadas.
*/

int numDec;
static int w = 3; //tamanho da janela de comparação.
static int l=4; //largura da área de busca
static int a=1; //altura da area de busca

int ntap = 24;
int esquerda=0;
int direita=0;

String nameOut;

public Estereo(ImagePlus left, ImagePlus right){
  createFilter();
  imagePlusRight = right;
  imagePlusLeft = left;
  numDec = 4; // numero de vezes que a imagem e decimada
  createImage = new CreateImage();
  iconeL = new Multivector[numDec+1][][];
  iconeR = new Multivector[numDec+1][][];
  disparidadeCol = new int[numDec+1][][];
  disparidadeLin = new int[numDec+1][][];
  iconeL[0] = createImage.convertIPtoCliffordMatriz(imagePlusLeft);
  iconeR[0] = createImage.convertIPtoCliffordMatriz(imagePlusRight);
}

public Estereo(String args[]){
  numDec = 4; // numero de vezes que a imagem e decimada
  createFilter();
  createImage = new CreateImage();
  iconeL = new Multivector[numDec+1][][];
  iconeR = new Multivector[numDec+1][][];
  disparidadeCol = new int[numDec+1][][];
  disparidadeLin = new int[numDec+1][][];

  ImagePlus imagePlusLY = new ImagePlus(args[0]);
  imagePlusLY.show();
  ImagePlus imagePlusLU = new ImagePlus(args[1]);
```

```
    imagePlusLU.show();
    ImagePlus imagePlusLV = new ImagePlus(args[2]);
    imagePlusLV.show();
    ImagePlus imagePlusRY = new ImagePlus(args[3]);
    imagePlusRY.show();
    ImagePlus imagePlusRU = new ImagePlus(args[4]);
    imagePlusRU.show();
    ImagePlus imagePlusRV = new ImagePlus(args[5]);
    imagePlusRV.show();

    iconeL[0] = createImage.
        convertYUVtoClifford(imagePlusLY,imagePlusLU,imagePlusLV);
    iconeR[0] = createImage.
        convertYUVtoClifford(imagePlusRY,imagePlusRU,imagePlusRV);

    imagePlusLY.close();
    imagePlusLU.close();
    imagePlusLV.close();
    imagePlusRY.close();
    imagePlusRU.close();
    imagePlusRV.close();

    nameOut = args[6];
    execute();
  }

  public Estereo(ImagePlus left, ImagePlus right, String nameOut){
    createFilter();
    imagePlusRight = right;
    imagePlusLeft = left;
    this.nameOut = nameOut;
    numDec = 4; // numero de vezes que a imagem e decimada
    createImage = new CreateImage();
    iconeL = new Multivector[numDec+1][][];
    iconeR = new Multivector[numDec+1][][];
    disparidadeCol = new int[numDec+1][][];
    disparidadeLin = new int[numDec+1][][];
    iconeL[0] = createImage.convertIPtoCliffordMatriz(imagePlusLeft);
    iconeR[0] = createImage.convertIPtoCliffordMatriz(imagePlusRight);
    right.close();
    left.close();
    execute();
  }

  public static void main(String args[]){
    String path[] = {"CARROSSEL/","TRENO/","RHINO/",
                     "FLAMINGO/","FISH/","WATER/" };
```

```
int nImage = 6;
String nfile;
int i = 0;
int k = 1;

/*  Estereoscopia do modelo YUV
for (; k < path.length; k++) {
  for (; i < nImage; i+=5) {
    if(i < 10)
      nfile = "00"+i;
    else
      nfile = "0"+i;
    String arg[] = {"images/"+path[k]+"LEFT/Y"+nfile+".tif",
                "images/"+path[k]+"LEFT/U"+nfile+".tif",
                "images/"+path[k]+"LEFT/V"+nfile+".tif",
                "images/"+path[k]+"RIGHT/Y"+nfile+".tif",
                "images/"+path[k]+"RIGHT/U"+nfile+".tif",
                "images/"+path[k]+"RIGHT/V"+nfile+".tif",
                "images/"+path[k]+"DYUV3"+nfile+".tif"};
    new Estereo(arg);
  }
  i = 0;
}
*/

/*  Estereoscopia do modelo RGB
for (; k <  path.length; k++) {
  for (i =0; i < nImage; i++) {
    if(i < 10)
      nfile = "00"+i;
    else
      nfile = "0"+i;
    ImagePlus left = new ImagePlus("images/"
                        +path[k]+"LEFT/L"+nfile+".tif");
    left.show();
    ImagePlus right = new ImagePlus("images/"+path[k]
                        +"RIGHT/R"+nfile+".tif");
    right.show();
    new Estereo( left, right,"images/"
                            +path[k]+"DC"+nfile+".tif");
  }
  i = 0;
  nImage = 10;
}
*/

nImage = 10;
```

```
    k=0;

    for (i = k; i < path.length; i++) {
      String arg[]={path[i],String.valueOf(nImage)};
      ReconsPart.main(arg);
      ErroQuadraticoMedio.main(arg);
    }

  }

  public void execute(){
    long tempInicial = System.currentTimeMillis();
    Thread um = new Thread(){public void run(){downSample(iconeL[0], 1);}};
    um.start();
    Thread dois = new Thread(){public void run(){downSample(iconeR[0], 2);}};
    dois.start();

    while(um.isAlive() || dois.isAlive());

    upSample();
    IJ.run("PNG...");
    long tempFinal = System.currentTimeMillis();

    float dif = (tempFinal - tempInicial);
    float minu = (float) ((dif/1000.0)/60.0);

    System.out.println("Tempo de execução "+minu+" m");
  }

  public void mapaDisparidade(int i, int j, int mc, int mr){
    Multivector sum;
    double sumAux;
    double sumin;
    int r,c,h,k;
    int n=-w;
    int s=w;
    int e=w;
    int o=-w;
    int m1=-a,m2=a;// -1 1 variação na altura da busca
    int l1=-l,l2=l;// -4 4 variação na largura da busca

    if(j+o<0)o=-j;            /*limita entro i bordi ricerca a sinistra*/
    if((j+e)>mc-1)e=mc-j-1;   /*limita entro i bordi ricerca a destra*/
    if(i+n<0)n=-i;            /*limita entro i bordi ricerca in alto*/
    if((i+s)>mr-1)s=mr-i-1;   /*limita entro i bordi ricerca in basso*/
    if(i+m1+n<0)m1=-i-n;
    if((i+m2+s)>mr-1)m2=mr-i-1-s;
```

```
    if(j+l1+o<0)l1=-j-o;
    if(j+l2+e>mc-1)l2=mc-j-1-e;

    sumin = 260000;

    for(r=m1;r≤m2;++r)
      for(c=l1;c≤l2;++c){
        sum = new Multivector();
        for (h=n;h≤s;h++)
          for (k=o;k≤e;k++){
            sum = sum.add(ABS(iconeL[numDec][i+r+h][j+c+k].subtract(
                          iconeR[numDec][i+h][j+k])));
          }
          sumAux = sum.norm_e();
          if(sumAux < sumin){
            sumin = sumAux;
            //System.out.printf(" i=%d j=%d sumin=%f ", i, j, sumin);
            disparidadeCol[numDec][i][j]=c;
            disparidadeLin[numDec][i][j]=r;
          }
      }

  }

  public void mapaDisparidadeLR(int i, int j, int mc, int mr){
    Multivector sum;
    double sumAux;
    double sumin;
    int r,c,h,k;
    int n=-w;
    int s=w;
    int e=w;
    int o=-w;
    int m1=-a,m2=a;// -1 1 variação na altura da busca
    int l1=-l,l2=l;// -4 4 variação na largura da busca

    if(j+o<0)o=-j;
    if((j+e)>mc-1)e=mc-j-1;
    if(i+n<0)n=-i;
    if((i+s)>mr-1)s=mr-i-1;
    if(i+m1+n<0)m1=-i-n;
    if((i+m2+s)>mr-1)m2=mr-i-1-s;
    if(j+l1+o<0)l1=-j-o;
    if(j+l2+e>mc-1)l2=mc-j-1-e;

    sumin = 260000;
```

```
  for(r=m1;r≤m2;r++)
    for(c=l1;c≤l2;c++){
      sum = new Multivector();
      for (h=n;h≤s;h++)
        for (k=o;k≤e;k++){
          sum = sum.add(ABS(iconeR[numDec][i+r+h][j+c+k]
                      .subtract(iconeL[numDec][i+h][j+k])));
        }
        //suml = (float)aux.norm_e();
        sumAux = sum.norm_e();
        if(sumAux < sumin){
          sumin = sumAux;
          //System.out.printf(" i=%d j=%d sumin=%f ", i, j, sumin);
          disparidadeColLR[i][j]=c;
          disparidadeLinLR[i][j]=r;
        }
     }
}

private Multivector ABS(Multivector value) {

  for (Object blade : value.getBlades())
    ((BasisBlade) blade).scale = Math.abs(((BasisBlade) blade).scale);

  return value;

}

/**
 * @param i
 * @param j
 * @param nivel
 * @param mc
 * @param mr
 */

private double disparidade(int iL, int jL, int iR, int jR, int nivel, int mc, int mr

  int h,k;
  int n=-w;
  int s=w;
  int e=w;
  int o=-w;
  int w2 = (2*w+1);

  double finestra = 1./(double)(w2*w2);
```

```
  if ( iL+n<0 )
    n=-iL;
  if ( (iR+s)>mr-1 )
    s=mr-iR-1;
  if ( iR+n<0 )
    n=-iR;
  if ( (iL+s)>mr-1 )
    s=mr-iL-1;
  if ( (jR+e)>mc-1 )
    e=mc-jR-1;
  if ( jL+o<0 )
    o=-jL;
  if ( (jL+e)>mc-1 )
    e=mc-jL-1;
  if ( jR+o<0 )
    o=-jR;

  Multivector suma = new Multivector();

  for (h=n;h≤s;h++)
    for (k=o;k≤e;k++)
      suma = suma.add(ABS(iconeL[nivel][iL+h][jL+k]
                  .subtract(iconeR[nivel][iR+h][jR+k])));

  return suma.norm_e()*finestra;

}

/**
 * Calcula a disparidade das linhas
 * @param lin = linha da matriz a ser verificada
 * @param col = coluna da matriz a ser verificada
 * @param nivel = nivel da imagem a ser utilizado
 */
public void disparidadeLin(int i, int j, int nivel){

  int mc = iconeL[nivel][0].length;
  int mr = iconeL[nivel].length;
  int mdx, mdy;

  double  v[] = new double[3] ;

  int c1=-1,c2=1;
  if(j+c1<0)c1=0;
  if(j+c2>mc-1)c2=0;
  int jc1 = j+c1;
  int jc2 = j+c2;
```

```
    mdy = (disparidadeCol[nivel][i][jc1]+disparidadeCol[nivel][i][jc2])/2;
    mdx = (disparidadeLin[nivel][i][jc1]+disparidadeLin[nivel][i][jc2])/2;

    v[0] = disparidade(i + mdx, j + mdy, i, j, nivel, mc, mr);
    v[1] = disparidade(i+disparidadeLin[nivel][i][jc1],
                jc1+disparidadeCol[nivel][i][jc1], i, j, nivel, mc, mr);
    v[2] = disparidade(i+disparidadeLin[nivel][i][jc2],
                jc2+disparidadeCol[nivel][i][jc2], i, j, nivel, mc, mr);

    double vmin = v[0];
    int idmin = 0;

    for (int k = 1; k < v.length; k++) {
      if(v[k]<vmin){
        idmin = k;
        vmin = v[k];
      }
    }

    switch (idmin) {
      case 0:
        disparidadeLin[nivel][i][j]=mdx;
        disparidadeCol[nivel][i][j]=mdy;
        break;

      case 1:
        disparidadeCol[nivel][i][j]=disparidadeCol[nivel][i][jc1]+c1;
        disparidadeLin[nivel][i][j]=disparidadeLin[nivel][i][jc1];
        break;

      case 2:
        disparidadeCol[nivel][i][j]=disparidadeCol[nivel][i][jc2]+c2;
        disparidadeLin[nivel][i][j]=disparidadeLin[nivel][i][jc2];
        break;

      default:
        disparidadeLin[nivel][i][j]=mdx;
        disparidadeCol[nivel][i][j]=mdy;
        break;

    }

  }

/**
 * Calcula a disparidade das colunas
```

```
* @param lin = linha da matriz a ser verificada
* @param col = coluna da matriz a ser verificada
* @param nivel = nivel da imagem a ser utilizado
*/

public void disparidadeCol(int i, int j, int nivel){
  int mc = iconeL[nivel][0].length;
  int mr = iconeL[nivel].length;
  int mdx, mdy;
  double  v[] = new double[3];
  int r1=-1,r2=1;

  if(i+r1<0)r1=0;
  if(i+r2>mr-1)r2=0;

  int ir1 = i+r1;
  int ir2 = i+r2;

  mdx= (disparidadeLin[nivel][ir1][j]+disparidadeLin[nivel][ir2][j])/2;
  mdy= (disparidadeCol[nivel][ir1][j]+disparidadeCol[nivel][ir2][j])/2;

  v[0] = disparidade(i + mdx,j + mdy, i, j, nivel, mc, mr);
  v[1] = disparidade(ir1+disparidadeLin[nivel][ir1][j] ,
          j+disparidadeCol[nivel][ir1][j], i, j, nivel, mc, mr);
  v[2] = disparidade(ir2+disparidadeLin[nivel][ir2][j] ,
          j+disparidadeCol[nivel][ir2][j] , i, j, nivel, mc, mr);

  double vmin = v[0];
  int idmin = 0;

  for (int k = 1; k < v.length; k++) {
    if(v[k]<vmin){
      idmin = k;
      vmin = v[k];
    }
  }

  switch (idmin) {
    case 0:
      disparidadeLin[nivel][i][j]=mdx;
      disparidadeCol[nivel][i][j]=mdy;
      break;

    case 1:
      disparidadeLin[nivel][i][j]=disparidadeLin[nivel][ir1][j]+r1;
      disparidadeCol[nivel][i][j]=disparidadeCol[nivel][ir1][j];
      break;
```

```
      case 2:
        disparidadeLin[nivel][i][j]=disparidadeLin[nivel][ir2][j]+r2;
        disparidadeCol[nivel][i][j]=disparidadeCol[nivel][ir2][j];
        break;

      default:
        disparidadeLin[nivel][i][j]=mdx;
        disparidadeCol[nivel][i][j]=mdy;
        break;

    }

}

/**
 * Calcula a disparidade central
 * @param lin = linha da matriz a ser verificada
 * @param col = coluna da matriz a ser verificada
 * @param nivel = nivel da imagem a ser utilizado
 */

public void disparidadeCentral(int i, int j, int nivel){
  int mc = iconeL[nivel][0].length;
  int mr = iconeL[nivel].length;
  int mdx, mdy, dx=0, dy=0;
  double  v[] = new double[5];
  int r1=-1,r2=1;
  int c1=-1,c2=1;

  if(i+r1<0)r1=0;
  if(i+r2>mr-1)r2=mr-i-1;
  if(i+r2<0)r2=0;
  if(j+c1<0)c1=0;
  if(j+c2>mc-1)c2=mc-j-1;
  if (j+c2<0)c2=0;
  int ir1 =i+r1;
  int ir2 =i+r2;
  int jc1 =j+c1;
  int jc2 =j+c2;

  for (int r=r1;r≤r2;r=r+2)
    for (int c=c1;c≤c2;c=c+2) {
      dx+=disparidadeLin[nivel][i+r][j+c];
      dy+=disparidadeCol[nivel][i+r][j+c];
    }
```

```
mdx = (disparidadeLin[nivel][i][j+c1] + disparidadeLin[nivel][i][j+c2])/2;
mdy = (disparidadeCol[nivel][i][j+c1] + disparidadeCol[nivel][i][j+c2])/2;
mdx = dx/4;
mdy = dy/4;

v[0] = disparidade(i + mdx, j + mdy, i, j, nivel, mc, mr);
v[1] = disparidade(i+r1+disparidadeLin[nivel][i+r1][j+c1],
            j+c1+disparidadeCol[nivel][i+r1][j+c1], i, j, nivel, mc, mr);
v[2] = disparidade(i+r1+disparidadeLin[nivel][i+r1][j+c2],
            j+c2+disparidadeCol[nivel][i+r1][j+c2], i, j, nivel, mc, mr);
v[3] = disparidade(i+r2+disparidadeLin[nivel][i+r2][j+c1],
            j+c1+disparidadeCol[nivel][i+r2][j+c1], i, j, nivel, mc, mr);
v[4] = disparidade(i+r2+disparidadeLin[nivel][i+r2][j+c2],
            j+c2+disparidadeCol[nivel][i+r2][j+c2], i, j, nivel, mc, mr);

double vmin = v[0];
int idmin = 0;

for (int k = 1; k < 5; k++) {
  if(v[k]<vmin){
    idmin = k;
    vmin = v[k];
  }
}

switch (idmin) {

  case 0:
    disparidadeLin[nivel][i][j]=mdx;
    disparidadeCol[nivel][i][j]=mdy;
    break;

  case 1:
    disparidadeLin[nivel][i][j]= disparidadeLin[nivel][ir1][jc1]+r1;
    disparidadeCol[nivel][i][j]= disparidadeCol[nivel][ir1][jc1]+c1;
    break;

  case 2:
    disparidadeLin[nivel][i][j]=disparidadeLin[nivel][ir1][jc2]+r1;
    disparidadeCol[nivel][i][j]=disparidadeCol[nivel][ir1][jc2]+c2;
    break;

  case 3:
    disparidadeLin[nivel][i][j]=disparidadeLin[nivel][ir2][jc1]+r2;
    disparidadeCol[nivel][i][j]=disparidadeCol[nivel][ir2][jc1]+c1;
    break;
```

```
      case 4:
        disparidadeLin[nivel][i][j]=disparidadeLin[nivel][ir2][jc2]+r2;
        disparidadeCol[nivel][i][j]=disparidadeCol[nivel][ir2][jc2]+c2;
        break;

      default:
        disparidadeLin[nivel][i][j]=mdx;
        disparidadeCol[nivel][i][j]=mdy;
        break;

    }

  }

  public void upSample() {
    System.out.print("\r Upsample 0");
    int mr = iconeL[numDec].length;
    int mc = iconeL[numDec][0].length;
    disparidadeCol[numDec] = new int[mr][mc];
    disparidadeLin[numDec] = new int[mr][mc];
    disparidadeColLR = new int[mr][mc];
    disparidadeLinLR = new int[mr][mc];

    int i, j;
    for (i=0; i<mr; i++)
      for (j=0; j<mc; j++)
        mapaDisparidade(i, j, mc, mr);

      /*for (i=0; i<mr; i++)
          for (j=0; j<mc; j++)
            mapaDisparidadeLR(i, j, mc, mr);

    int conflit= 0, conflit2=0, cobid=0;

    System.out.println(" O CARA "+disparidadeCol[numDec][8][40]);

    for ( i=0; i<mr; i++)
      for ( j=0; j<mc; j++){
        try {
          if ( disparidadeColLR[i][j]==((-1)*disparidadeCol[numDec][i][j-disparidadeCo
          else {
            conflit2 ++;//-disparidadeLinLR[i][j]
            if( Math.abs(disparidadeColLR[i][j]) <
              Math.abs(disparidadeCol[numDec][i-disparidadeLinLR[i][j]][j-disparidadeC
              && ((j-disparidadeColLR[i][j]-disparidadeColLR[i][j]) ≥0)
              && ((j-disparidadeColLR[i][j]-disparidadeColLR[i][j]) ≤45)) {
              conflit++;
```

```
            disparidadeCol[numDec][i][j-disparidadeColLR[i][j]]= (-1)*disparidadeCol
            disparidadeLin[numDec][i][j-disparidadeLinLR[i][j]]= (-1)*disparidadeLin
          }
        }
      } catch (Exception e) {
        //System.out.println(" i-disparidadeLinLR[i][j]"+(i-disparidadeLinLR[i][j])
        //System.out.println(" j-disparidadeColLR[i][j]"+(j-disparidadeColLR[i][j])
        //e.printStackTrace();
        }
    }

System.out.println(" O CARA denovo "+disparidadeCol[numDec][8][40]);
createImage.convertMatrizInttoImage(disparidadeCol[numDec], 1,"Disparidade Horizon
IJ.run("PNG...");
//System.out.println(" Conflit "+conflit+ " confli2 "+ conflit2+" conbid "+cobid);
*/

for(int k =0; k < numDec; k++) {
  System.out.print("\r Upsample "+(k+1));

  int oldMr = mr;
  int oldMc = mc;
  int nivel = numDec-1-k;
  int nivelK = numDec-k;

  mc = 2*mc;
  mr = 2*mr;

  disparidadeCol[nivel] =  new int[mr][mc];
  disparidadeLin[nivel] =  new int[mr][mc];

  int ii;

  for( i=oldMr;--i≥0;){
    ii = i*2;
    for( j=oldMc;--j≥0;){
      disparidadeCol[nivel][ii][j*2] = disparidadeCol[nivelK][i][j]*2;
      disparidadeLin[nivel][ii][j*2] = disparidadeLin[nivelK][i][j]*2;
    }
  }

  for( i=0;i<mr;i+=2)
    for( j=0;j<mc;j+=2)
      aggiorna(i,j,nivel,mc,mr);

  for( i=0;i<mr;i+=2)
    for( j=1;j<mc;j+=2)
```

```
          disparidadeLin(i, j, nivel);

      for( i=1;i<mr;i+=2)
        for( j=0;j<mc;j+=2)
          disparidadeCol(i, j, nivel);

      for( i=1;i<mr;i+=2)
        for( j=1;j<mc;j+=2)
          disparidadeCentral(i, j, nivel);

      //ImagePlus out = createImage.convertMatrizInttoImage(disparidadeCol[nivel], 1,n
      //out.show();
      //IJ.run("PNG...");
    }

    ImagePlus out = createImage.convertMatrizInttoImage(disparidadeCol[0], 1,nameOut);
    FileSaver fs = new FileSaver(out);
    fs.saveAsTiff(nameOut);
    out.close();
  }

  public void aggiorna(int i, int j, int nivel, int mc, int mr){
    int r,c,dx,dy;
    int h1=-1,h2=1;
    int k1=-1,k2=1;
    double sumin=2600;
    double sum;

    dx=disparidadeLin[nivel][i][j];
    dy=disparidadeCol[nivel][i][j];

    if(i+dx+h1<0)h1=0;
    if(i+dx+h2>mr-1)h2=0;
    if(j+dy+k1<0)k1=0;
    if(j+dy+k2>mc-1)k2=0;

    for (r=h1;r≤h2;++r)
      for (c=k1;c≤k2;++c) {
        sum=disparidade((i+dx+r),(j+dy+c),i,j,nivel,mc,mr);
        if (sum < sumin) {
          sumin=sum;
          disparidadeCol[nivel][i][j]=(dy+c);
          disparidadeLin[nivel][i][j]=(dx+r);
        }
      }
    return;
  }
```

```
public void downSample(Multivector[][] image, int side) {
  int width = image[0].length;
  int height = image.length;
  Multivector aux[][];
  int ii,jj;

  for (int k = 0; k < numDec; k++) {
    convolve1DVector(image, width, height);
    System.out.print("\r DownSample "+(numDec-k)+"Image"+side+" h "+height +" W "+ w
    ii = height;
    jj = width;
    width = width /2;
    height = height/2;
    aux = new Multivector[height][width];

    for (int i = 0; i < ii; i += 2) {
      for (int j = 0; j < jj; j += 2)
        aux[i/2][j/2] = image[i][j];
    }

    image = aux;
    if(side==1){
      iconeL[k+1]=aux.clone();
      /*ImagePlus out = createImage.multivectorToImageYUV(iconeL[k+1], "IconeL"+widt
      out.show();
      IJ.run("PNG...");*/
    }

    else{
      iconeR[k+1]=aux.clone();
      /*ImagePlus out = createImage.multivectorToImageYUV(iconeR[k+1], "IconeR"+widt
      out.show();
      IJ.run("PNG...");*/
    }
  }
}

public void convolve1DVector(Multivector image[][], int width, int height) {
  int mr = height;
  int mc = width;
  Multivector temp[] = new Multivector[width];
  Multivector aux = new Multivector();
  ArrayList<BasisBlade> listAjusArrayList = new ArrayList<BasisBlade>(3);
  listAjusArrayList.add(new BasisBlade((byte) 1, 0.5));
  listAjusArrayList.add(new BasisBlade((byte) 2, 0.5));
  listAjusArrayList.add(new BasisBlade((byte) 4, 0.5));
```

```
    Multivector ajuste = new Multivector(listAjusArrayList);
    int i , j;
    for ( i = 0; i < mr; i++) {
      for ( j = 0; j < mc; j++){
        aux = convoloriz(i, j, image, width);
        aux = aux.add(ajuste);
        checkValue(aux);
        temp[j] = aux;
      }
      for ( j = 0; j < mc; j++)
        image[i][j] = temp[j];
    }
    temp = new Multivector[height];
    for ( j = 0; j < mc; j++ ) {
      for ( i = 0; i < mr; i++){
        aux = convolvert(i, j, image, height);
        aux = aux.add(ajuste);
        checkValue(aux);
        temp[i] = aux;
      }
      for ( i = 0; i < mr; i++)
        image[i][j] = temp[i];
    }
  }

  /**
   * Checa se os valores do Multivector estão entre 0 e 255;
   * @param value
   */
  private void checkValue(Multivector value){
    for( Object blade : value.getBlades()){
      if(((BasisBlade) blade).scale<0)
        ((BasisBlade) blade).scale = 0;

      else if (((BasisBlade) blade).scale > 255)
        ((BasisBlade) blade).scale = 255;

    }
  }

  /**
   * Compoe o multivector e um unico valor com as soma
   * do valor absoluto de cada espaco;
   * @param value
   */

  private double sumofValue(Multivector value){
```

```
    double sum = 0;

    for (Object blade : value.getBlades())
      sum += Math.abs(((BasisBlade) blade).scale);
    return sum;
  }

  /*---------------Convoloriz-------------------------*/
  /* Funzione che esegue la convoluzione orizzontale */
  /*--------------------------------------------------*/
  public Multivector convoloriz(int nlinha, int ncol, Multivector image[][], int width
    Multivector sum = new Multivector();
    int r = 0;
    int j = 0;
    int mc = width;
    int auxNtap = ntap / 2;

    for (r = -auxNtap ; r < auxNtap + 1; r++) {
      j = ncol + r;
      if (j < 0)
      j = 0;
      if (j > mc - 1)
      j = mc - 1;

      sum = sum.add(filterFirL[r + auxNtap].gp(image[nlinha][j]));

    }
    checkValue(sum);
    return sum;
  }

  /*---------------Convolvert-------------------------*/
  public Multivector convolvert(int nlinha, int ncol, Multivector image[][], int heigh
  {
    Multivector sum = new Multivector();
    int r = 0;
    int i = 0;

    int mr = height;
    int auxNtap = ntap/2;
    for (r = -auxNtap; r < auxNtap + 1; r++) {
      i = nlinha + r;
      if (i < 0)
        i = 0;
      if (i > mr - 1)
        i = mr - 1;
      sum = sum.add(filterFirL[r + auxNtap].gp(image[i][ncol]));
```

```
    }
    checkValue(sum);
    return sum;
  }

  private void createFilter(){
    double filter[] = { -0.005615, -0.005127, 0.011230,
                0.012451, -0.017090, -0.023193, 0.021484,
                0.047119, -0.025635, -0.095703, 0.029785,
                0.314697, 0.471191, 0.314697, 0.029785, -0.095703,
                -0.025635, 0.047119, 0.021484, -0.023193,
                -0.017090, 0.012451, 0.011230, -0.005127,
                -0.005615 };
    filterFirL = new Multivector[filter.length];
    for (int i = filter.length; --i ≥0;) {
      filterFirL[i] = new Multivector(new BasisBlade( 0, filter[i]));
    }
  }
```

APÊNDICE

B

Códigos Fonte Erro Quadrático Médio

```
import ij.ImagePlus;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import javax.swing.JTextArea;

public class ErroQuadraticoMedio{

  private static double EQM = 0.0;
  private static double ps=0.0;
  static double somaEQM = 0.0;
  private static int height;
  private static int width;

  public ErroQuadraticoMedio(String image, String imageRec){

    ImagePlus imagePlusOriginal = new ImagePlus(image);
    imagePlusOriginal.show();
    ImagePlus imagePlusReconstruida = new ImagePlus(imageRec);
    imagePlusReconstruida.show();
    height = imagePlusOriginal.getHeight();
    width = imagePlusOriginal.getWidth();

    somaEQM=0.0;
    ps = 0.0;
    EQM = 0.0;

    for (int i = 0; i < height; i++) {
```

```
      for (int j = 0; j < width; j++) {
        somaEQM += eqm(imagePlusOriginal.getPixel(i,j),imagePlusReconstruida.getPixel
      }
    }

    somaEQM = somaEQM/(width*height*3.0); // ruído
    //somaEQM = Math.sqrt(somaEQM);
    EQM = ps/(somaEQM*width*height*3.0);
    //somaEQM = Math.sqrt(somaEQM);
    imagePlusOriginal.close();
    imagePlusReconstruida.close();
    System.out.println(" EQM= "+somaEQM);

    //EQM = ps/somaEQM; //relação sinal ruido
    //EQM=ps/(somaEQM*width*height);
    System.out.printf("  SNR=%20.15f dB\n",10*(Math.log10(EQM)));
    EQM = 10*(Math.log10(EQM));

  }

  private static double eqm(int v1[], int v2[]){
    double res=0.0;
    ps += Math.pow(v1[0],2);
    ps += Math.pow(v1[1],2);
    ps += Math.pow(v1[2],2);

    res= Math.pow((v1[0]-v2[0]),2); //R
    res+= Math.pow((v1[1]-v2[1]),2); //G
    res+= Math.pow((v1[2]-v2[2]),2); //B

    res= Math.pow(v1[0]/(v1[0]-v2[0])/(width*height*3),2); //R
    res+= Math.pow(v1[1]/(v1[1]-v2[1])/(width*height*3),2); //G
    res+= Math.pow(v1[2]/(v1[2]-v2[2])/(width*height*3),2); //B

    return Math.sqrt(res);

  }

  public static void main(String []args){

    JTextArea text = new JTextArea();
    String nomeFile= null;
    String nfile;
    String path = args[0];
    int nImage = Integer.valueOf(args[1]);
    for (int i = 0; i < nImage; i++) {
      if(i < 10)
```

```
      nfile = "00"+i;
    else
      nfile = "0"+i;
    nomeFile = "images/"+path+"/RecDD4C"+nfile+".tif";
    new ErroQuadraticoMedio("images/"+path+"RIGHT/R"+nfile+".tif", nomeFile);
    text.append(nomeFile+" NSR "+EQM+" EQM "+somaEQM+System.getProperty("line.separa
  }

  FileOutputStream out;
  DataOutputStream p;

  try {
    out = new FileOutputStream("images/"+path+"EQMD4.txt");
    p = new DataOutputStream( out );
    p.writeBytes(text.getText());
    p.close();
  }
  catch (Exception e)
  {
    System.err.println ("Error writing to file");
  }
}
```